Roberto Ierusalimschy

PONTIFÍCIA UNIVERSIDADE CATÓLICA
DO RIO DE JANEIRO

Lua

- O que
- Onde
- Por que
- Como

O Que é Lua

- Mais uma linguagem de script
  - alguma similariedade com Perl, Python, Tcl
- Uma linguagem de descrição de dados
  - anterior a XML
- Uma linguagem de extensão estensível
  - ênfase em comunicação inter-linguagens
  - enfatiza desenvolvimento em múltiplas linguagens

O Que é Lua

- Única linguagem desenvolvida fora do eixo EUA/Europa/Japão a ser adotada mundialmente
  - Ruby é (a única) do Japão

Onde Lua é Desenvolvida

- Desenvolvida na PUC-Rio
  - desde 1993
- Início modesto, para uso interno
  - expansão lenta e gradual
- "Comitê" de três pessoas
  - Roberto Ierusalimschy, Luiz H. de Figueiredo, Waldemar Celes

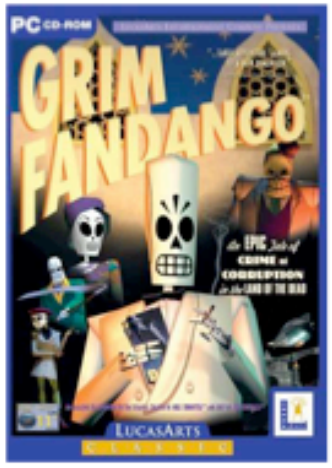
Onde Lua é Usada

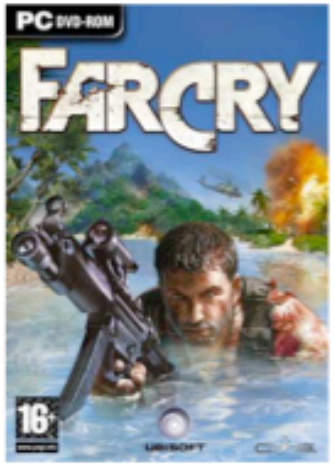
- Nicho em jogos
- pesquisa informal feita pelo site **`gamedev.net`** (set. 2003) mostrou Lua como a linguagem mais usada para script em jogos

Lua em Jogos

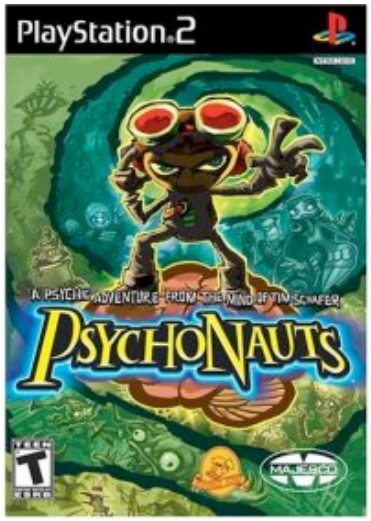
- "It is easy to see why Lua is rapidly becoming the de facto standard for game scripting." *Artificial Intelligence for Games*, Morgan Kaufmann, 2006.
- "It's quite possible that game developers will look back at the 2000s as the decade of Lua"., *Game Programming Gems 5*, Charles River Media, 2005.

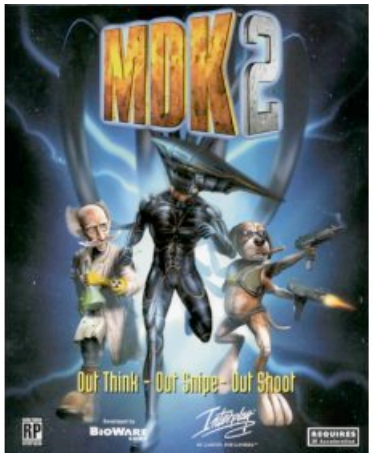
Alguns Jogos que usam Lua

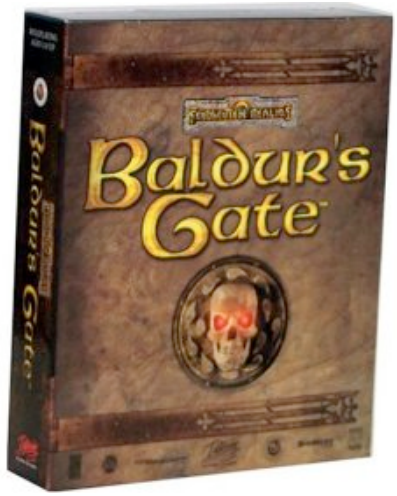


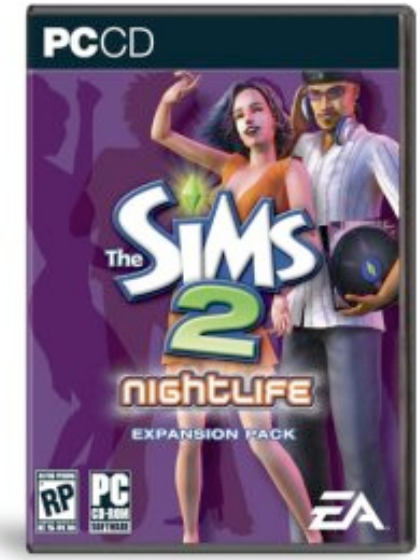


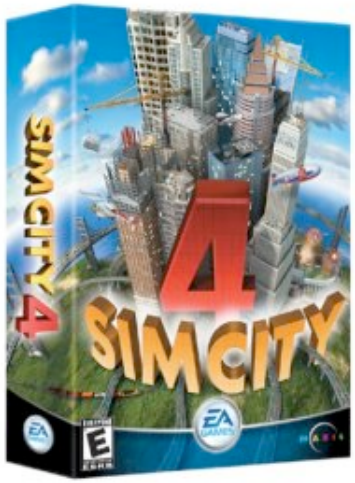
Alguns Jogos que usam Lua (2)



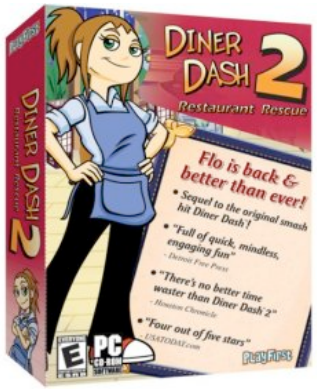


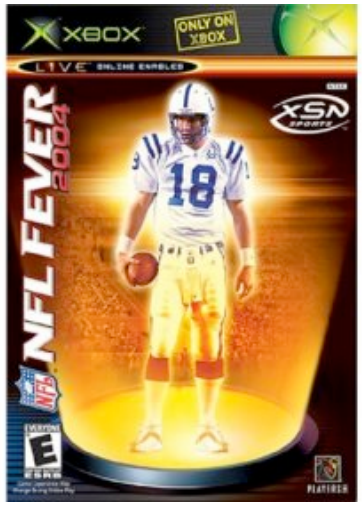


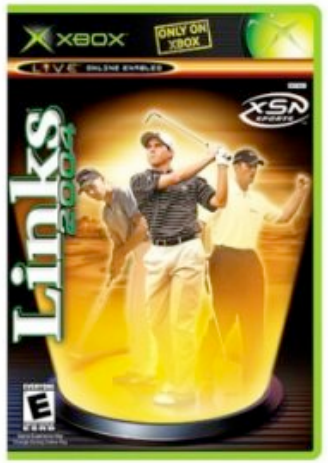


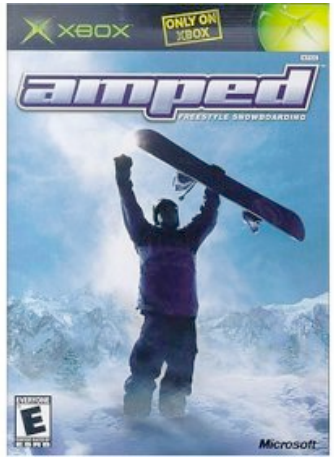


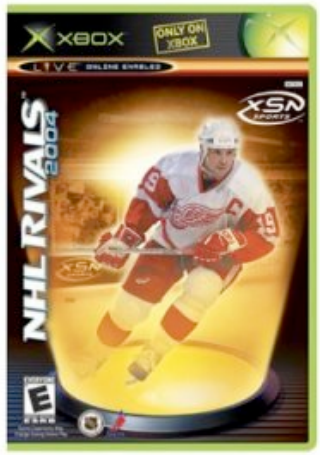
Alguns Jogos que usam Lua (3)











Mas Existem Muitas Outras Aplicações Importantes

"Adobe Lightroom is ‘the complete, elegant environment for the art and craft of digital photography from raw capture to creative output’. Over 40% of Adobe Lightroom is written in Lua.” Mark Hamburg (Adobe Fellow)

E Muitos Outros Usos

- firmware para impressoras (Olivetty), analisador de protocolos (Wireshark), biblioteca para GIS (INPE), monitoramento remoto (Omnitronix), pós-produção de filmes (Digital Fusion, eyeon), servidores Web (RealTimeLogic), etc.

Porque Lua

- portabilidade
- simplicidade
- pequeno tamanho
- “acoplabilidade”
- eficiência

Portabilidade

- Roda em praticamente todas as plataformas que já ouvimos falar
  - Unix, Windows, Windows CE, Symbian, BREW, hardware dedicado, Palm, PSII, etc.
- Escrita em ANSI C ∩ ANSI C++
  - evita `#ifdefs`
  - evita pontos obscuros do padrão

Simplicidade

- Um único tipo de estrutura de dados
  - tabelas
- Um único tipo numérico
  - tipicamente double
- Mecanismos ao invés de políticas
  - e.g., orientação a objetos

Pequeno Tamanho

- Menos de 200K
- Núcleo + bibliotecas
  - interface bem definida
  - núcleo com menos de 100K
  - bibliotecas independentes (e removíveis)

Acoplabilidade

- Lua é uma biblioteca C
- API simples e bem definida
  - tipos simples
  - operações primitivas
  - modelo de pilha
- Acoplada em C/C++, Java, Fortran, C#, Perl, Ruby, Ada, etc.

Eficiência

- Benchmarks independentes mostram Lua entre as mais rápidas no grupo de linguagens interpretadas com tipagem dinâmica
- Mistura de algumas técnicas especiais e simplicidade

Como é Lua

- Sintaxe convencional

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

```
function fact (n)
  local f = 1
  for i=2,n do
    f = f * i
  end
  return f
end
```

Tabelas

- Arrays associativos
  - qualquer valor como chave
- semântica referencial
- Único mecanismo de estruturação de dados
- Construtores

```
{}
{x = 5, y = 10}
{“Sun”, “Mon”, “Tue”}
{[exp1] = exp2, [exp3] = exp4}
```

Estruturas de Dados

- Tabelas implementam maioria das estruturas de forma simples e eficiente
- estruturas: açucar sintático `t.x` para `t["x"]`:

```
t = {}
t.x = 10
t.y = 20
print(t.x, t.y)
print(t["x"], t["y"])
```

Estruturas de Dados (2)

- Arrays: inteiros como índices

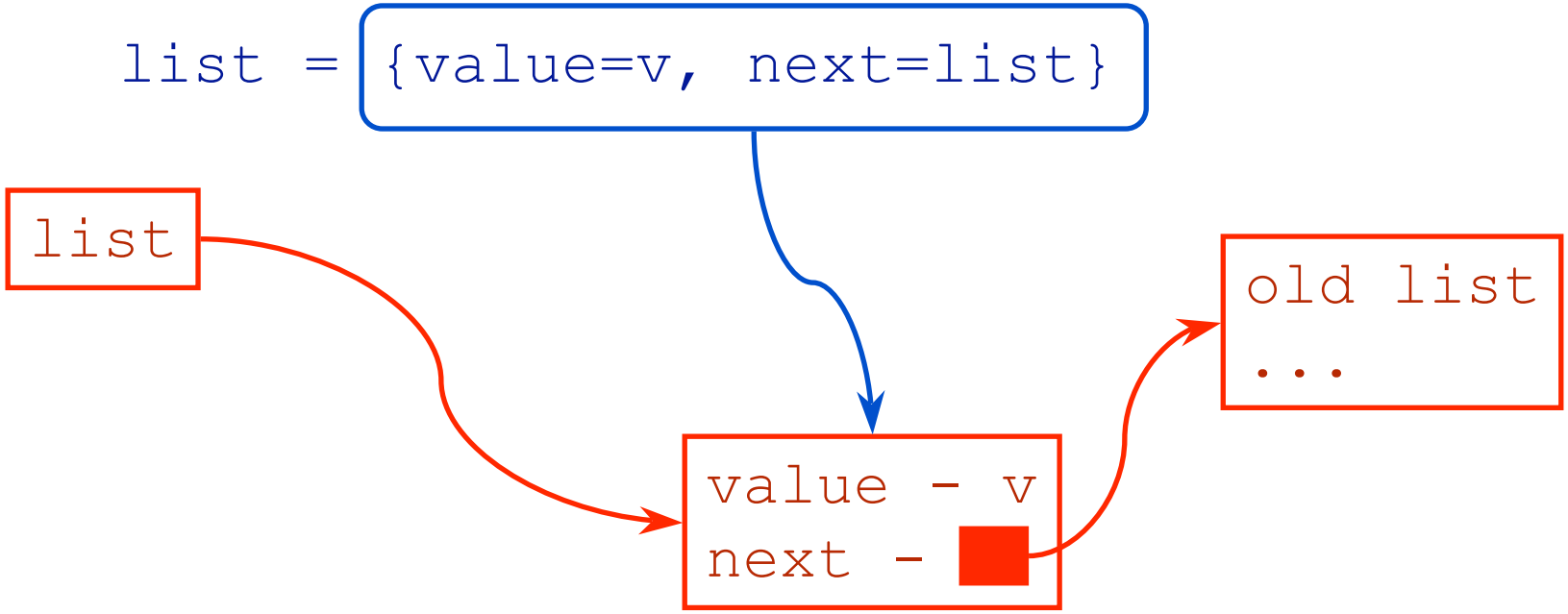
```
a = {}
for i=1,n do a[i] = 0 end
```

- Conjuntos: elementos como índices

```
t = {}
t[x] = true       -- t = t ∪ {x}
if t[x] then      -- x ∈ t?
  ...
```

Listas Encadeadas

- Tabelas são *objetos*, criados dinamicamente



Strings

- Valores imutáveis
- Sem limite de tamanho
  - muitos programas lêem arquivo inteiro de entrada de uma só vez
- Podem guardar valores binários arbitrários
- Biblioteca (externa) para pattern-matching

Funções

- Valores de 1ª classe

```
function inc (x)
  return x+1
end
```


```
inc = function (x)
        return x+1
      end
```

- Múltiplos resultados

```
a, b = f()
print(f())
```

```
function f()
  return 1,2
end
```

Funções

- Escopo Léxico

```
function make_counter (x)
  return funcion ()
           x = x + 1
           return x
         end
end

c1 = make_counter(10)
print(c1()) --> 11
print(c1()) --> 12
```

Módulos

- Tabelas povoadas por funções

```
require “math”
print(math.sqrt(10))
```

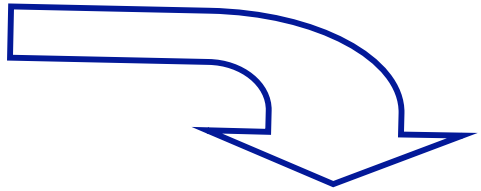
- Várias facilidades vem de graça
  - sub-módulos
  - nomes locais

```
local m = require “math”
print(m.sqrt(20))
local f = m.sqrt
print(f(10))
```

Objetos

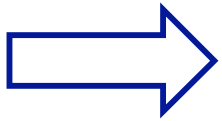
- Funções de 1ª classe + tabelas ≈ objetos
- Açucar sintático para métodos
  - cuida de *self*

```
function a:foo (x)
  ...
end
```



```
a.foo = function (self,x)
  ...
end
```

```
a:foo(x)
```



```
a.foo(a,x)
```

Delegação

- Quando *a* delega de *b*, qualquer campo ausente em *a* é pego de *b*
  - `a[k]` resulta em `(a[k] or b[k])`
- Permite objetos baseados em protótipo e em classes
- Permite herança simples

Co-rotinas

- Lua implementa co-rotinas assimétricas, de 1ª classe, “stackfull”
- (Podemos implementar `call/cc`)
- Podemos implementar multi-threading cooperativo (não preemptivo)

Reflexividade

- Introspecção
  - função `type`
  - percorrimento de tabelas

```
function clone (t)
  local new = {}
  for k,v in pairs(t) do
    new[k] = v
  end
  return new
end
```

- Acesso à tabela de globais

```
for n in pairs(_G) do
  print(n)
end
```

Reflexividade (2)

- Chamadas dinâmicas

```
t[1] = a; t[2] = b;
f(unpack(t))
```

- Interface de depuração
  - pilha de execução
  - variáveis locais
  - linha corrente

API Lua-C

- Biblioteca reentrante
- Casamento de impedância:
  - tipagem dinâmica x estática
  - gerenciamento de memória automático x manual
- Usa uma pilha para troca de valores entre linguagens

API Lua-C (2)

- Carrega código Lua
  - em arquivos, strings, etc.
- Chama funções Lua
- Manipula dados em Lua
- Registra funções C para serem chamadas pelo código Lua

Interpretador Lua Básico


```
#include <lua.h>
#include <lauxlib.h>
#include <lualib.h>

int main (int argc, char **argv) {
  lua_State *L = luaL_newstate();
  luaL_openlibs(L);
  luaL_loadfile(L, argv[1]);
  lua_call(L, 0, 0);
  return 0;
}
```

Comunicação Lua - C

- Toda troca de dados via pilha de valores Lua

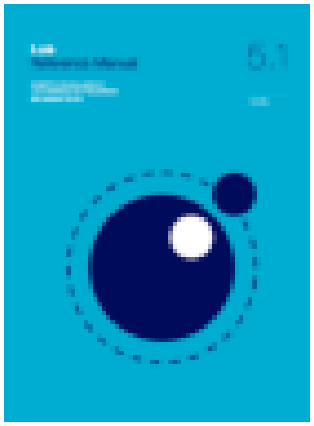
```
/* calling f("hello", 4.5) */
lua_getglobal(L, "f");
lua_pushstring(L, "hello");
lua_pushnumber(L, 4.5);
lua_call(L, 2, 1);
if (lua_isnumber(L, -1))
  printf("%f\n", lua_getnumber(L, -1));
```

Funções C

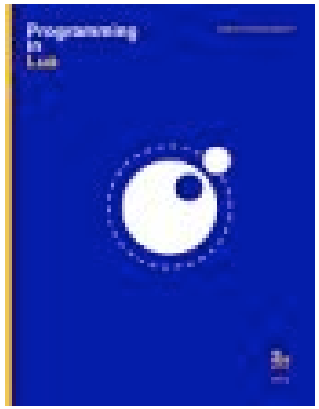

```
static int l_sqrt (lua_State *L) {
  double n = luaL_checknumber(L, 1);
  lua_pushnumber(L, sqrt(n));
  return 1;  /* number of results */
}
```

```
lua_pushcfunction(L, l_sqrt);
lua_setglobal(L, "sqrt");
```

Livros

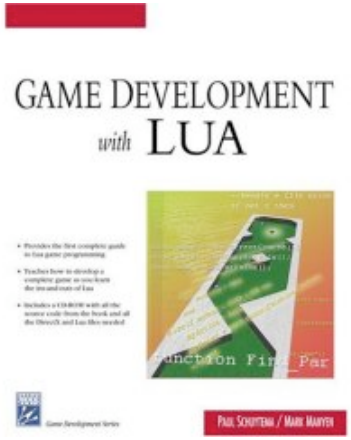


Lua 5.1 Reference Manual
Lua.org, 2006
ISBN 85-903798-3-3



Programming in Lua, 2nd ed.
Lua.org, 2006
ISBN 85-903798-2-5

Livros



Game Development with Lua
Charles River Media, 2005
ISBN 1-58450-404-8

Programmieren mit Lua
Open Source Press, 2006
ISBN 3-937514-22-8

Beginning Lua Programming
Wrox, 2007
ISBN 0-470-06917-1

Para Saber Mais...

www.lua.org