Learn Lua in $X$ minutes

Roberto Ierusalimschy
What is Lua

• Yet another dynamic language
  • not totally unlike Perl, Python, Tcl

• A scripting language
  • emphasis in inter-language communication

• Particular set of goals:
  • embedability
  • portability
  • simplicity
  • small size
Embedability

- Provided as a library
- Simple API
  - simple types
  - low-level operations
  - stack model
- Embedded in C/C++, Java, Fortran, C#, Perl, Ruby, Ada, etc.
Portability

- Runs on most machines we ever heard of
  - Unix, Windows, Windows CE, Symbian, embedded hardware, Palm, Sony PSP, etc.
- Written in ANSI C ∩ ANSI C++
  - avoids #ifdefs
  - avoids dark corners of the standard
Simplicity

• Just one data structure
  • tables
• Complete manual with 100 pages
• Mechanisms instead of policies
Small Size

• Less than 200K
  • less than 20K lines of C code

• Core + libraries
  • clear interface
  • core has less than 100K
  • easy to remove libraries
Lua is also quite Efficient

• Several independent benchmarks show Lua as the most efficient in the realm of dynamically-typed interpreted languages
• Efficient in real code, too
• Smart implementation
  • register-based virtual machine
  • novel algorithm for tables
  • small and simple (!)
Uses of Lua

• Embedded systems
  • internet switches, robots, keyboards (Logitech G15), LCDs

• Scripting
  • Metaplace, nmap, Wireshark, Snort

• Programming
  • Adobe Photoshop Lightroom

• Niche in games
How Is Lua

• Conventional syntax

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

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

• Associative arrays
  • any value as key
• Variables store references to tables, not tables
• Only data-structuring mechanism
  • easily implements arrays, records, sets, etc.
Table Constructors

- Syntax to create tables

```lua
{}  
{x = 5, y = 10}  
{"Sun", "Mon", "Tue"}  
{10, 20, 30, n = 3}  
{["+"] = "add", ["-"] = "sub"}  

{{x=10.5, y=13.4},  
{x=12.4, y=13.4},  
... }
```
Data Structures

• Tables implement most data structures in a simple and efficient way
• records: syntactical sugar \texttt{t.x} for \texttt{t["x"]}:

```lua
local t = {}  
t.x = 10  
t.y = 20  
print(t.x, t.y)  
print(t["x"], t["y"])
```
Data Structures (2)

- Arrays: integers as indices
  
  ```plaintext
  a = {}
  for i=1,n do a[i] = 0 end
  a[1000000000] = 1
  ```

- Sets: elements as indices
  
  ```plaintext
  t = {}
  t[x] = true        -- t = t ∪ {x}
  if t[x] then        -- x ∈ t?
    ...
  ```
Linked Lists

• Tables are *objects*, created dynamically

\[ \text{list} = \{\text{value}=v, \text{next}=\text{list}\} \]
Data Description

book{
    author = "F. P. Brooks",
    title = "The Mythical Man-Month",
    year = 1975
}

book{
    author = "Brian Kernighan & Rob Pike",
    title = "The Practice of Programming",
    year = 1999
}

...
Functions

• First-class Values

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

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

• Multiple returns

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

```lua
a, b = f()
prompt(f())
{f()}
```
Functions

• Lexical Scoping

```lua
function newcounter (x)
    return funcion ()
        x = x + 1
    return x
end
end

c1 = newcounter(10)
c2 = newcounter(20)
print(c1()) --> 11
print(c2()) --> 21
print(c1()) --> 12
```
Modules

- Tables populated with functions

```latex
require "math"
print(math.sqrt(10))
print(type(math))       --> table
print(type(math.sqrt))  --> function
```
Modules as Tables

• Several facilities come for free
  • submodules
  • local names

```lua
local m = require "math"
print(m.sqrt(20))
local f = m.sqrt
print(f(10))
```
Objects

- First-class functions + tables ≈ objects
- Syntactical sugar for methods
  - handles *self*

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

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

a:foo(x) → a.foo(a, x)
point = {x = 10, y = 0}
function point:move (dx, dy)
    self.x = self.x + dx
    self.y = self.y + dy
end

point:move(5, 5)
print(point.x, point.y)

point.move(point, 4, 2)
print(point.x, point.y)
Delegation

- When table $a$ delegates from table $b$, any field absent in $a$ is got from $b$
  - $a[k]$ becomes $a[k]$ or $b[k]$
- Allows prototype-based and class-based objects
- Allows single inheritance
Delegation in Lua: example

Point = {x = 0, y = 0}
function Point:move (dx, dy)
    <as before>

function Point:new (o)
    setmetatable(o, {__index = self})
    return o
end

p = Point:new{x = 5}
p:move(10, 10)
print(p.x, p.y)
Delegation in Lua: example

Point = {x = 0, y = 0}
function Point:move (dx, dy)
  <as before>

function Point:new (o)
  setmetatable(o, {__index = self})
  return o
end

p = Point:new{x = 5}
p:move(10, 10)
print(p.x, p.y)
delegation trick
Active Delegation

• When \textit{a} delegates from a function \textit{b}, any field absent in \textit{a} is got from calling \textit{b}
  • \(a[k]\) becomes \(a[k] \text{ or } b(a,k)\)
• Allows all kinds of inheritance
• Also implements proxies and similar structures
Coroutines

- Lua implements asymmetric, first-class, "stackfull" coroutines
- (We can implement call/cc on top of them)
- We can implement cooperative (non-preemptive) multithreading on top of them
Reflexive Facilities

- Introspection
  - function **type**
  - table traversal

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

- Access to global table

```lua
for n in pairs(_G) do
  print(n)
end
```
Reflexive Facilities (2)

- Dynamic calls

- Debug interface
  - execution stack
  - local variables
  - current line

\[ t[1] = a; t[2] = b; f(unpack(t)) \]
Lua-C API

• Reentrant library
• Impedance mismatch:
  • dynamic x static typing
  • automatic x manual memory management
• Uses a stack for inter-language communication
Lua-C API (2)

- Load Lua code
  - in files, strings, etc.
- Call Lua functions
- Manipulate Lua data
- Register of C functions to be called by Lua code
Basic Lua Interpreter

```c
#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);
    lua_close(L);
    return 0;
}
```
Communication Lua - C

• All data exchange through a stack of Lua values

/* 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));
C Functions

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");
Books

- Beginning Lua Programming
  Wrox, 2007

- World of Warcraft Programming
  Wiley, 2008

- Lua 5.1 Reference Manual
  Lua.org, 2006

- Game Development with Lua
  Charles River Media, 2005
Books

Programming in Lua, 2nd edition
Lua.org, 2006

Programmieren mit Lua
Open Source Press, 2006

程序设计：第2版
PHEI, 2008

프로그래밍 루아
Insight, 2007
To Know More...

www.lua.org