Lua: 20 Years

Roberto Ierusalimschy
Luiz Henrique de Figueiredo
Waldemar Celes
1992-1993: The Beginning
1992: Tecgraf

Partnership between PUC-Rio and Petrobras (the Brazilian Oil Company)
1992: Tecgraf

• Two projects using “little languages”

DEL, for data entry

PGM, to visualize geologic profiles
DEL
Data Entry Language

- form definition
- parameter list
- types and default values

```plaintext
:e gasket "gasket properties"
mat s  # material
d f 0  # distance
y f 0  # settlement stress
t i 1  # facing type

:p gasket.m>30
gasket.m<3000
gasket.y>335.8
gasket.y<2576.8
```
SOL
Simple Object Language

- data description language
- not totally unlike XML (with DTD)
- BibTeX-like syntax

```
type @track {x:number, y:number=23, z}
type @line {t:@track=@track{x=8}, z:number*}

-- create an object 't1', of type 'track'
t1 = @track {y=9, x=10, z="hi!”}

l = @line {t=@track{x=t1.y, y=t1.x}, z=[2,3,4]}
```
1992: Tecgraf

- Both DEL & PGM shared several limitations
  - decision-making facilities
  - arithmetic expressions
  - abstraction mechanisms
1993

- Roberto (PGM), Luiz (DEL) and Waldemar (PGM) got together to find a common solution to their common problems...
What we needed?

• A “generic configuration language”
• A “complete” language
• Easily embeddable
• Portable
• Non-intimidating syntax
• As simple as possible
As we were giving up Sol,

a friend suggested a new name...
...and Lua was born
Lua 1.0 (1993)

- Called 1.0 a posteriori
- *The simplest thing that could possibly work*
- Standard implementation
  - precompiler with yacc/lex
  - opcodes for a stack-based virtual machine
- Less than 5000 lines of C code
Lua 1.0 (1993)

- Requirements:
  - simple, portable, extensible, embeddable, small

- Expectations:
  - to solve our problems with PGM and DEL
  - fulfilled our expectations: both DEL and PGM used Lua successfully

- It was a big success in Tecgraf
Soon, several projects at Tecgraf were using Lua
Lua 1.1 (1994)

- New users brought new demands
  - several small improvements
  - mainly for performance
- Reference manual
- Well-defined and well-documented C API
License

- First public distribution
  - ftp
  - free for academic uses, but not free for commercial uses
  - after one year, only one unfruitful contact

- Later versions with free license
  - open-source license
  - not “open-source decision making”
How was Lua 1.0?

- Not that different from Sol...

```lua
t1 = @track{x = 10.3, y = 25.9, title = "depth"}
```
How was Lua 1.0?

- But quite different...

```lua
function track (t)
    if not t.x then t.x = 0.0 end
    if type(t.x) ~= "number" then
        error("invalid 'x' value")
    end
    if type(t.y) ~= "number" then
        error("invalid 'y' value")
    end
end
```

t1 = @track{x = 10.3, y = 25.9, title = "depth"}

•
Tables in Lua 1.0

· The sole data structure
  ▪ still is

· Constructors:

```
t = @(100)       -- empty
```
```
t = @[a, b, c]   -- list
```
```
t = @{x = 10, y = 20} -- record
```
```
t = @foo{x=10, y = 20}
```
Tables in Lua 2.1 (1995)

- Any value as index
  - not only numbers and strings

- Simplification
  - Only one constructor + syntactic sugar:

  ```
  t = {}               -- empty
  t = {a, b, c}        -- list
  t = {x = 10, y = 20} -- record
  t = foo({x=10, y = 20})
  ```
Lua 2.1 - 2.5 (Feb 95 - Nov 96)

• Fallbacks
  • minimum mechanism to get the label “OO inside”

• External precompiler
  • faster load for large programs (metafiles)

• Debug facilities
  • basic mechanisms for external debuggers

• Pattern matching
Tables everywhere

- As Lua evolved, the use of tables only increased
- All data structures
  - arrays, records, sets, lists, etc.
  - objects
  - modules
- Tables also used for several internal structures
  - global variables, metamethods, references (in the API)
This is the first public release of Lua.

* What is Lua?
Lua is a simple, yet powerful, language for extending applications. Lua has been developed by TeCGraf, the Computer Graphics Technology Group of PUC-Rio, the Catholic University of Rio de Janeiro, Brazil. Dozens of industrial products developed by TeCGraf use Lua.
[...]

International Exposure
International Exposure

• First home page in 1995
  ▪ http://www.inf.puc-rio.br/~roberto/lua

• e-mail contact with “far-away” users

• Beginning of 1997 - discussion list
  ▪ to allow users to answer users' questions
  ▪ end of 1997 - more than 100 subscribers, should we try a newsgroup?
International Exposure

- **Jun 1996 - paper in S:P&E**

- **Dec 1996 - article in Dr. Dobb's**
Hi there...

After reading the Dr. Dobbs article on Lua I was very eager to check it out, and so far it has exceeded my expectations in every way! It's elegance and simplicity astound me. Congratulations on developing such a well-thought out language.

Some background: I am working on an adventure game for the LucasArts Entertainment Co., and I want to try replacing our older adventure game scripting language, SCUMM, with Lua.
Lucas Arts, 1998:  
First AAA game to use Lua
Scripting in Grim Fandango

“[The engine] doesn't know anything about adventure games, or talking, or puzzles, or anything else that makes Grim Fandango the game it is. It just knows how to render a set from data that it's loaded and draw characters in that set.

[...] The real heroes in the development of Grim Fandango were the scripters. They wrote everything from how to respond to the controls to dialogs to camera scripts to door scripts to the in-game menus and options screens. [...] “A TREMENDOUS amount of this game is written in Lua. The engine, including the Lua interpreter, is really just a small part of the finished product.”

Bret Mogilefsky
What happened next

- Game of the year...almost
  - Half-Life relegated us the Adventure Game of the Year
- GDC 1999 (2000?)
  - Panel discussion of scripting languages
    - Rob Huebner on embedding Java
    - Kevin Bruner on interpreted C++
    - Seamus McNally on not using a scripting language
  - 200 miserable people
  - “Or you could just use Lua...”
  - Furious scribbling
Enter door to LUA Bar
1998 - 2002

- User base grows outside Brazil
  - shift from inside PUC to outside
- Development of a virtual community
- Writing of PiL
- Several changes in the language
  - support for larger programs
  - functional facilities
  - several incompatible changes between versions
Lua 3.1 (1998)

- **Functional features**
  - syntax for anonymous, nested functions
  - since Lua 1.0, `function f ...` was sugar for `f = function ...`, except that the latter was not valid syntax!

```lua
foreach(t, function (k, v)
  print(k, v)
end)
```

```lua
button.action = function ... end
```
Lexical scoping

• Functional features

• No simple and efficient way to implement lexical scoping
  • on-the-fly compilation with no intermediate representation + activation records in a stack
  • hindered earlier adoption of nested functions
Upvalues

- “a form of proper lexical scoping”
- The frozen value of an external local variable inside a nested function
- Trick somewhat similar to Java demand for final when building nested classes
- Special syntax to avoid misunderstandings

```lua
function f (x)
    return function () return %x end
end
```
In Aug 1998, Cameron Laird wrote in SunWorld:

“Its user base is also small; there might be only a few tens of thousands of Lua programmers in the world. They're very fond of this language, though, and the imminent explosion of ubiquitous embedded processing (computers in your car, in your plumbing, and in your kitchen appliances) can only work in favor of Lua.”
In Aug 1998, Cameron Laird wrote in SunWorld:

“Its user base is also small; there might be only a few tens of thousands of Lua programmers in the world. They're very fond of this language, though, and the imminent explosion of ubiquitous embedded processing (computers in your car, in your plumbing, and in your kitchen appliances) can only work in favor of Lua.”

For us, this “small base” was much larger than we could have imagined!
Lua 3.2 (1999)

- Multithreading?
  - for Web servers
Lua 3.2

- Multithreading?
- Problems with multithreading
  - (preemption + shared memory)
  - not portable
  - no one can write correct programs when \( a = a + 1 \) is non-deterministic
  - core mechanisms originally proposed to OS programming, not to “normal people”
  - almost impossible to debug
 Lua 3.2

- Multithreading?
- Multiple “Lua processes”
  - multiple independent states in an application
  - no shared memory
- Would require major change in the API
  - each function should get the state as an extra argument
  - instead, a single C global variable in the code points to the running state
  - Extra API functions set the running state
Lua 4.0 (2000)

- Major change in the API
  - all functions got a new parameter (the state)
  - no more C global variables in the code
  - libraries should not use C globals, too
  - concurrent C threads can each have its own state

- We took the opportunity and made several other improvements in the API
  - stack oriented
2001

• Several appearances in Brazilian press
• March 2001, new site: www.lua.org
  • gift from a user (Jim Mathies)
• Few months later, lua-users.org
  • kept by Lua users (thanks to John Belmonte)
• Big plans for 4.1
Plans for Lua 4.1

- Multithreading?
  - multiple characters in games
Plans for Lua 4.1

- Multithreading?
- Coroutines!
  - portable implementation
  - deterministic semantics
  - coroutines + scheduler = non-preemptive multithreading
  - could be used as a basis for multithreading for those that really wanted it
Plans for Lua 4.1

- New implementation for tables
  - store array part in an actual array
- New implementation for upvales
  - allowed “true” lexical scoping!
- New register-based virtual machine
- Tags replaced by metatables
  - regular tables that store *metamethods* (old *tag methods*) for the object
Plans for Lua 4.1

- New implementation for tables
  - store array part in an actual array
- New implementation for upvales
  - allowed “true” lexical scoping!
- New register-based virtual machine
- Tags replaced by metatables
  - regular tables that store *metamethods* (old *tag methods*) for the object

Too much for a minor version...
Lua 5.0 (2003)

- Coroutines
- Lexical scoping
- Register-based virtual machine, new implementation for tables
- Metatables, boolean type, weak tables, proper tail calls, ...
- Module system
Lua 5

• Dec 2003 - 1st edition of “Programming in Lua”
• Mar 2004 - roundtable about Lua at GDC
• Jul 2005 - 1st Lua workshop
  • San Jose, CA (sponsored by Adobe)
• Mar 2006 - 2nd edition of “Programming in Lua”
Lua 5.1 (2006)

- Incremental garbage collector
  - demand from games
- Better support for modules
  - more policies
  - functions to help following “good practice”
- Support for dynamic C libraries
  - not portable!
  - the mother of all (non portable) libraries
• 2007: III ACM History of Programming Languages.

• 2009: Masterminds of Programming: Conversations with the Creators of Major Programming Languages. O'Reilly Media.

Lua 5.2 (2011)

- Yieldable pcall and metamethods
- Emergency collection
- New lexical scheme for global variables (_ENV)
- Ephemeron tables
- bitlib library
- Finalizers for tables
- Light C functions
Lua Today
Embedded Systems

Samsung (TVs), Cisco (routers), Logitech (keyboards), Olivetti (printers), Océ (printers), Ginga (Brazilian TV middleware), Verison (set-top boxes), Texas Instruments (calculators Nspire), Huawei (cell phones), Sierra Wireless (M2M devices), Mercedes-Benzs (cars), ...
Adobe Lightroom
more than one million
lines of Lua code
Slashdot, Feb 1, 2012:

“Wikipedia Chooses Lua As Its New Template Language”

Wired, March 19, 2013:

“Meet Wikipedia, the Encyclopedia Anyone Can Code”

“As of this weekend, anyone on Earth can use Lua [...] to build material on Wikipedia and its many sister sites, such as Wikiquote and Wiktionary.”
Scripting the Internet of Things


October 2013: “Wind River Unveils Latest Software Platform for Internet of Things”
Lua in Games

- The Engine Survey (Mark DeLoura, 03/02/09, Gamasutra)
- What script languages are most people using?

![Bar chart showing Lua as the most popular script language, followed by Custom Language, C/C++ Variant, Unreal Script, and Python.](chart.png)
What is Next?

• Integers
  • 64-bit values in “large” machines
  • 32-bit integers and floats in “small” machines

• Extra libraries
  • basic UTF-8 support
  • struct

• Macros
  • how to do it?
Our “Principles”

- It is much easier to add a missing feature than to remove an excessive one
- Implementing is the easiest part of any new feature
  - documentation, maintenance, added complexity
  - it is very hard to anticipate all implications of a new feature
- Stuck to the standard
- Emphasis on embedding
Our “Principles”

• Mechanisms instead of policies
  • type definitions in Sol
  • delegation in Lua 2.1
  • coroutines
  • modules

• Effective way to avoid tough decisions

• This itself is a policy...