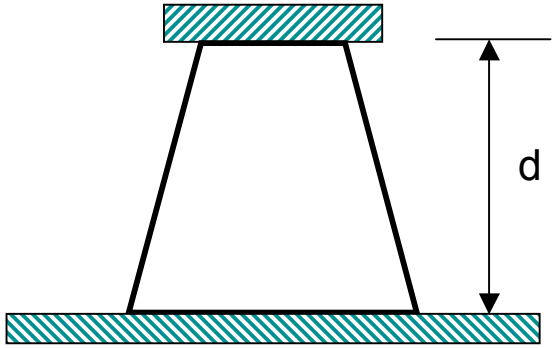
The Evolution of Lua

Waldemar Celes
Luiz Henrique de Figueiredo
Roberto Ierusalimschy

The Beginning

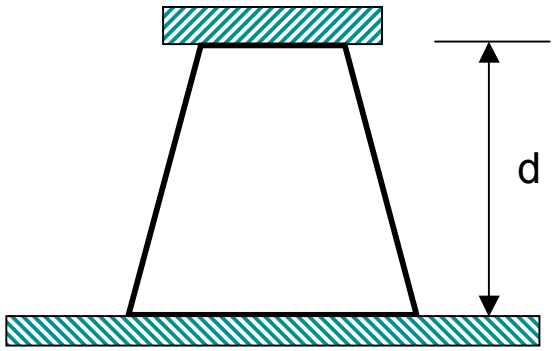
Data Entry Programs

- 1992: Tecgraf - partnership between PUC and PETROBRAS (CENPES)
- Graphical data entry for several simulation applications



DEL - Data Entry Language

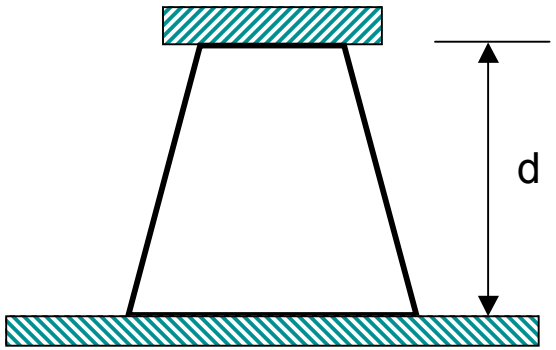
- Form definition
  - ⇨ parameter list
  - ⇨ types and default values




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

DEL limitations

- Input validation
- Conditional behavior
- Abstractions and basic arithmetic

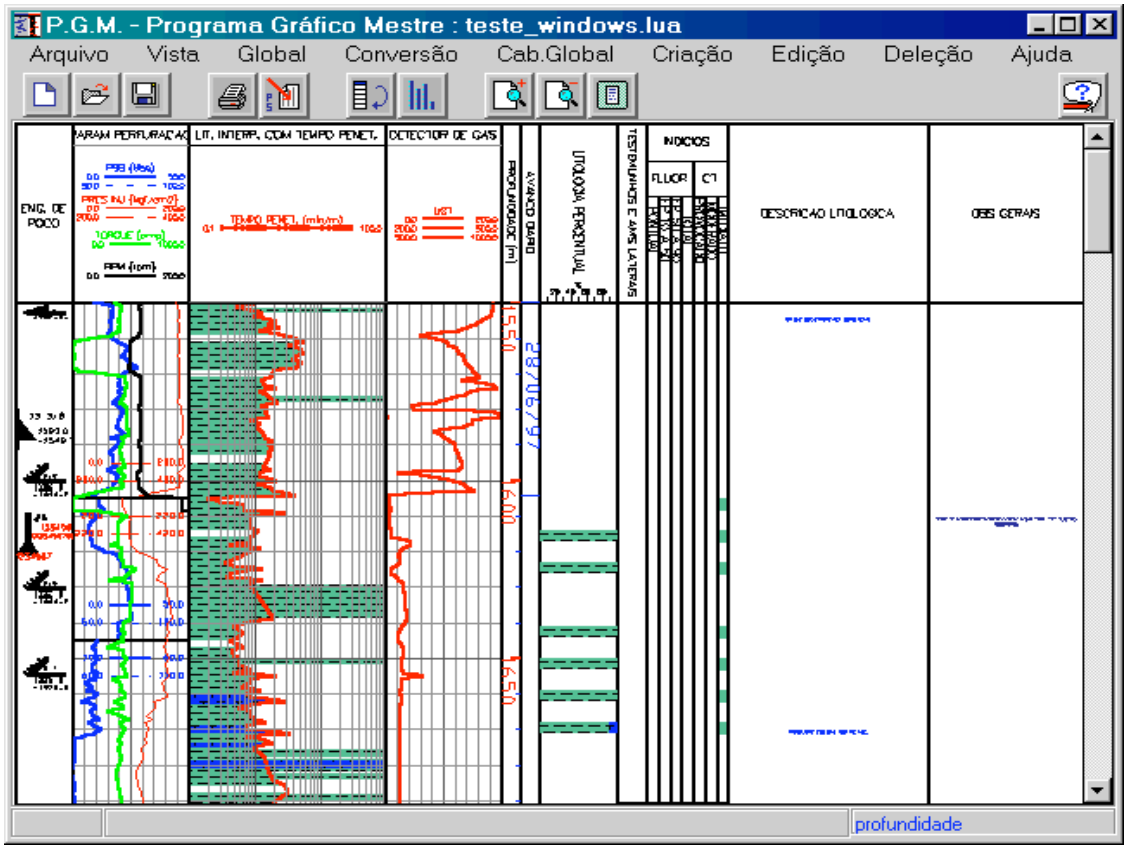



```
:e gasket "gasket properties"
mat     s               # material
d       f       0       # distance
y       f       0       # settlement stress
t       i       1       # facing type

:p gasket.m>30
gasket.m<3000
gasket.y>335.8
gasket.y<2576.8
```

Programa Gráfico Mestre

- 1993: another project with PETROBRAS
  - ⇨ configurable program to visualize geologic profiles



SOL
Simple Object Language

- Language to describe structured data
  - ⇨ not totally unlike XML
  - ⇨ syntax inspired by BibTeX

```
type @track {x:number, y:number=23, z}

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

-- creates 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] }
```

SOL limitations

- Stronger abstraction mechanisms
- Some procedural facilities

1993: Lua is Born

- Convergence of both languages
  - ⇨ procedural paradigm
  - ⇨ data-description mechanisms
- Powerful features
  - ⇨ function abstractions
  - ⇨ full arithmetic syntax
- Extensible extension language

Lua version 1.0

- Called 1.0 a posteriori
- *The simplest thing that could possibly work*
- Standard implementation, with yacc/lex
- Requirements:
  - ⇨ simple, portable, extensible, embedable, small

Lua 1.1 (1994)

- Faster
- First public distribution
  - ftp
- Free for academic uses, but not free for comercial uses

Lua 2

- Lua 2.1 (Feb 1995) - 2.5 (Nov 1996)
- Free license
- *Fallbacks*
  - ⇨ suport for OO programming
- Pattern matching
  - ⇨ Lua 2.5
- CGILua
  - ⇨ called HTMLLua (1995)

International Exposure

- First home page in 1995
  - ⇨ http://www.inf.puc-rio.br/~roberto/lua
- e-mail contact with far-away users
- June 1996 - paper in S:P&E
- Dez 1996 - paper in Dr. Dobb's
- Beginning of 1997 - discussion list
  - ⇨ end of 97 - more than 100 subscribers, should we try a newsgroup?

Lua 3

- Lua 3.0 (July 1997) - Lua 3.2 (July 1999)
- 1998, Lua logo
- 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.

Lua 4

- Lua 4.0 (Nov 2000 - March 2003)
- New API with `lua_State`
- Several appearances in Brazilian press
- March 2001, new site: www.lua.org
  - ⇨ thanks to Jim Mathies
- Few months later, users site: lua-users.org
- After two years, a single release 4.0.1
  - ⇨ less than 10 bugs
- Several plans for 4.1

Lua 5

- 5.0 (April 2003), 5.1 (Feb 2006)
- Coroutines, lexical scoping
- Register-based virtual machine
- New implementation for tables
- Modules
- Incremental garbage-collector

Another View of Lua Evolution

Evolution: Portability

- Stick to ANSI
  - ⇨ hard decision when we started
  - ⇨ Sun compiler was K&R
- Much improved
  - ⇨ first versions do not compile on Linux ;)
- Closely following ANSI C
  - ⇨ Lua and C compilers

Evolution: Portability

- General move from *conforming hosted implementation* to *conforming freestanding implementation* (in the core)
  - ⇨ no I/O in the core
  - ⇨ no use of files in the core
  - ⇨ user-provided memory-allocation mechanisms

Evolution: Portability

- But: loadlib deeply supported
- Module system
- `luaconf.h`
- Use of vararg and structs in `lua.h`
  - ⇨ only in restricted ways

Portability Evolution: Examples

- All kinds of problems with names
  - ⇨ `exp`, `size`, `Object`,
- Warnings
  - ⇨ no standard way to say "we know what we are doing"
- Compiler writers as language designers
  - ⇨ `tmpnam` in Linux, string functions in Windows

Portability Evolution: Examples


```
while(isalpha(*s++)) ...
```

The header <ctype.h> declares several functions useful for classifying and mapping characters. In all cases the argument is an int, the value of which shall be representable as an unsigned char or shall equal the value of the macro EOF.

```
for (; p >= base_pointer; p--) ...
```

union of pointers x pointer to union

Evolution: Embedability

- Portability
- Freestanding implementation
- Weak references
  - ⇨ non-lock references, weak tables
- Userdata
  - ⇨ from pointer to memory
  - ⇨ finalizers
  - ⇨ tags, metatables
- Independent states

Evolution: Simplicity

- Runs against all other aspects
  - ⇨ not always ;)
- Lua 1.1: API with 30 functions; 4000 lines of code
- Lua 5.1: API with 79 (core) + 36 (auxlib) functions; 12000 (core) + 5000 (libs) lines of code
  - ⇨ ~3x (core) + 1 (libs)

Evolution: Simplicity


```
/* Lua 1.1 */

int main (int argv, char **argc) {
  iolib_open();
  strlib_open();
  mathlib_open();
  lua_dofile(argv[1]);
  return 0;
}
```

Evolution: Simplicity


```
/* Lua 5.1 */

int main (int argv, char **argc) {
  lua_State *L = luaL_newstate();
  luaL_openlibs(L);
  if (luaL_loadfile(L, argv[1]) ||
      lua_pcall(L, 0, 0, 0))
    fprintf(stderr, "error: %s\n",
                    lua_tostring(L, -1));
  lua_close(L);
  return 0;
}
```

Evolution: Performance

Version	sieve	fibonacci	heapsort
Lua 1.0	1.8	2.2	
Lua 1.1	1.8	2.2	
Lua 2.1	0.8	4.1	
Lua 2.2	0.8	4.5	
Lua 2.4	0.8	4.4	0.5
Lua 2.5	0.7	3.3	0.4
Lua 3.0	0.7	3.8	0.5
Lua 3.1	0.8	3.7	0.5
Lua 3.2	0.7	3.5	0.5
Lua 4.0	0.6	3.7	0.4
Lua 5.0	0.4	3.0	0.3
Lua 5.1	0.4	3.3	0.3

Feature Evolution

Functions

- First-class values since Lua 1.0
- But quite different from functions in Lua 5.1

Functions in Lua 1.0

- Function definition assigned at compile time
- Must be a global name
- Cannot print function values

```
print(a())
function a() end

print(a)
function a.x() end
```

Functions in Lua 2.2

- Function definition is an assignment
- Function "name" may be a field
- Sugar for methods

```
function a.x[10]:m()
end
```

Functions in Lua 3

- Lua 3.0: vararg functions
- Lua 3.0: types `function` and `cfunction` unified
- Lua 3.1: anonymous functions with upvalues!

```
function createK (x)
  return function ()
           return %x
         end
end
```

Functions in Lua 5

- Full lexical scoping

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

- Proper tail calls
- Lua 5.1: new vararg expression

Chunks and Functions

- In Lua 2.4, chunks are functions internally
  - ⇨ debug interface could capture them illegally
- In Lua 2.5, chunks can return values
- In Lua 3.1, chunks are quite regular functions
  - ⇨ nesting
  - ⇨ local variables
- In Lua 5.0, "do" becomes "load" + "call"
  - ⇨ chunks are vararg functions

Error Messages

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

print(f(10))
```

Lua 1.0: without debug pragma

```
lua: unexpected type at conversion to number
```

Error Messages

```
$debug
function f(x)
  return x + y
end

print(f(10))
```

Lua 1.0: with debug pragma

```
lua: unexpected type at conversion to number
        in statement begining at line 3
                in function "f" of file "a"
        active stack
        -> function "f" of file "a"
```

Error Messages

```
$debug
function f(x)
  return x + y
end

print(f(10))
```

Lua 2.1:

```
lua: unexpected type at conversion to number
        active stack:
        -> function "f" at file "a":3
```

(Without pragma it is similar to Lua 1)

Error Messages

```
$debug
function f(x)
  return x + y
end

print(f(10))
```

Lua 3.0:

```
lua: unexpected type at arithmetic operation
Active Stack:
        function f at line 3 [in file a]
        main of a at line 6
```

("at line" only with pragmas)

Error Messages

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

print(f(10))
```

Lua 4.0: no more pragmas

```
a: attempt to perform arithmetic on global `y'
                                    (a nil value)
stack traceback:
   1:  function `f' at line 2 [file `a']
   2:  main of file `a' at line 5
```

Error Messages

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

print(f(10))
```

Lua 5.0:

```
a:2: attempt to perform arithmetic
                  on global `y' (a nil value)
stack traceback:
        a:2: in function `f'
        a:5: in main chunk
        [C]: ?
```

What are the costs of a feature?

Implementation is a small fraction of the cost of a new feature!

What are the costs of a feature?

- Documentation
  - ⇨ simple and precise description
  - ⇨ independent of implementation

- Testing and maintenance
  - ⇨ how to test all aspects
  - ⇨ more things to fail now and later

What are the costs of a feature?

- Conceptual integrity
  - ⇨ how the feature interacts with other features
  - ⇨ some features demand new facilities
- Impact on design space for future evolution
  - ⇨ a poor feature may stand in the way of a better one
- Impact on alternative implementations

Example: What is the cost of Multiple Returns?

- Documentation mostly about interation with other facilities
  - ⇨ “non local” documentation

- Conceptual integrity
  - ⇨ took long time to current design
    - ✧ f(g()) in 1.0-1.1; back in Lua 4.0
    - ✧ {f()} only in 5.0!

Example: What is the cost of Multiple Returns?

- Impact on design space for future evolution
  - ⇨ int return in C functions
  - ⇨ multiple values in resume-yield

- Impact on alternative implementations
  - ⇨ function stack size cannot be statically computed
  - ⇨ tail-call implementation
  - ⇨ implementations in virtual machines (e.g., JVM)

Example: What is the cost of Incremental GC?

- Practically no impact on documentation
- HUGE impact on testing
- Small impact on design space for future evolution
  - ⇨ finalizers, weak tables
- Big impact on alternative implementations
  - ⇨ several assumptions spread around the code

Example: What is the cost of `strsub`?

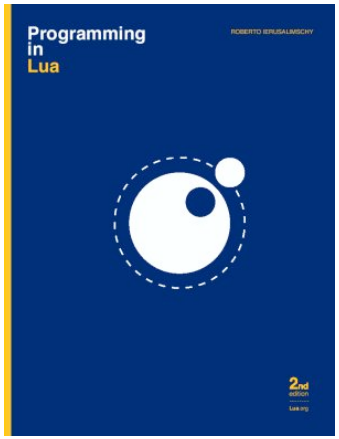
- Innocent-looking function in Lua 1.0
- Big impact on conceptual integrity
  - ⇨ particular way of interpreting string indices
- Set the tone for all other string-manipulation functions
- `string.find(s, "p")` x `string.match(s, "()p()")`
- Maybe Icon style would be better?

Lua Now

- Thirteen years
- More and more stable
  - ⇨ less and less unstable
- Still the same requirements
  - ⇨ simplicity, portability, embeadability, smallness
- Only language developed outside an industrialised country to achieve global relevance

Books

Programming in Lua
by Roberto Ierusalimschy.
Lua.org (2006)



Lua 5.1 Reference Manual
by Roberto Ierusalimschy,
Luiz H. de Figueiredo,
Waldemar Celes.
Lua.org (2006)

Game Development with Lua
by Paul Schuytema, Mark Manyen.
Charles River (2005)

Books

Programmieren mit Lua
by Roberto Ierusalimschy.
Open Souce Press (Set 2006)

Beginning Lua Programming
by Kurt Jung and Aaron Brown.
Wrox (Feb 5, 2007)

www.lua.org