Lua (programming language)

Lua (/ˈluːə/ *LOO-ə*, from Portuguese: *lua* [ˈlu.(w)ɐ] meaning *moon*; explicitly not "LUA"[1]) is a lightweight multi-paradigm programming language designed as a scripting language with extensible semantics as a primary goal. Lua is cross-platform since it is written in ANSI C,[1] and has a relatively simple C API.[2]

Lua ⎯⎯⎯ ⎯⎯⎯⎯

⎯⎯⎯ /lu / " Lua" (

)

	Lua
Paradigm	Multi-paradigm: scripting, imperative (procedural, prototype-based object-oriented), functional
Designed by	Roberto Ierusalimschy Waldemar Celes Luiz Henrique de Figueiredo
First appeared	1993; 22 years ago
Stable release	5.3.0 / 12 January 2015
Preview release	5.3.0 rc4 / 6 January 2015
Typing discipline	dynamic, strong, duck
Implementation language	C
OS	Cross-platform
License	MIT License
Filename extensions	.lua
Website	www.lua.org
Major implementations	
Lua, LuaJIT, LLVM-Lua, Lua Alchemy	
Dialects	
Metalua, Idle, GSL Shell	
Influenced by	
C++, CLU, Modula, Scheme, SNOBOL	
Influenced	
Io, GameMonkey, Squirrel, Falcon, MiniD, Julia	

History [edit]

Lua was created in 1993 by Roberto Ierusalimschy, Luiz Henrique de Figueiredo, and Waldemar Celes, members of the Computer Graphics Technology Group (Tecgraf) at the Pontifical Catholic University of Rio de Janeiro, in Brazil.

From 1977 until 1992, Brazil had a policy of strong trade barriers (called a market reserve) for computer hardware and software. In that atmosphere, Tecgraf's clients could not afford, either politically or financially, to buy customized software from abroad. Those reasons led Tecgraf to implement the basic tools it needed from scratch.[3]

Lua's historical "father and mother" were the data-description/configuration languages *SOL* (Simple Object Language) and *DEL* (data-entry language).[4] They had been independently developed at Tecgraf in 1992–1993 to add some flexibility into two different projects (both were interactive graphical programs for engineering applications at Petrobras company). There was a lack of any flow-control structures in SOL and DEL, and Petrobras felt a growing need to add full programming power to them.

As the language's authors wrote, in *The Evolution of Lua*:

> In 1993, the only real contender was Tcl, which had been explicitly designed to be embedded into applications. However, Tcl had unfamiliar syntax, did not offer good support for data description, and ran only on Unix platforms. We did not consider LISP or Scheme because of their unfriendly syntax. Python was still in its infancy. In the free, do-it-yourself atmosphere that then reigned in Tecgraf, it was quite natural that we should try to develop our own scripting language ... Because many potential users of the language were not professional programmers, the language should avoid cryptic syntax and semantics. The implementation of the new language should be highly portable, because Tecgraf's clients had a very diverse collection of computer platforms. Finally, since we expected that other Tecgraf products would also need to embed a scripting language, the new language should follow the example of SOL and be provided as a library with a C API.[3]

Lua 1.0 was designed in such a way that its object constructors, being then slightly different from the current light and flexible style, incorporated

the data-description syntax of SOL (hence the name Lua – *sol* is Portuguese for sun; *lua* is moon). Lua syntax for control structures was mostly borrowed from Modula (if, while, repeat/until), but also had taken influence from CLU (multiple assignments and multiple returns from function calls, as a simpler alternative to reference parameters or explicit pointers), C++ ("neat idea of allowing a local variable to be declared only where we need it"[3]), SNOBOL and AWK (associative arrays). In an article published in *Dr. Dobb's Journal*, Lua's creators also state that LISP and Scheme with their single, ubiquitous data structure mechanism (the list) were a major influence on their decision to develop the table as the primary data structure of Lua.[5]

Lua semantics have been increasingly influenced by Scheme over time,[3] especially with the introduction of anonymous functions and full lexical scoping.

Versions of Lua prior to version 5.0 were released under a license similar to the BSD license. From version 5.0 onwards, Lua has been licensed under the MIT License. Both are permissive free software licences and are almost identical.

Features [edit]

Lua is commonly described as a "multi-paradigm" language, providing a small set of general features that can be extended to fit different problem types, rather than providing a more complex and rigid specification to match a single paradigm. Lua, for instance, does not contain explicit support for inheritance, but allows it to be implemented with metatables. Similarly, Lua allows programmers to implement namespaces, classes, and other related features using its single table implementation; first-class functions allow the employment of many techniques from functional programming; and full lexical scoping allows fine-grained information hiding to enforce the principle of least privilege.

In general, Lua strives to provide flexible meta-features that can be extended as needed, rather than supply a feature-set specific to one programming paradigm. As a result, the base language is light — the full reference interpreter is only about 180 kB compiled[1] — and easily adaptable to a broad range of applications.

Lua is a dynamically typed language intended for use as an extension or scripting language, and is compact enough to fit on a variety of host platforms. It supports only a small number of atomic data structures such as boolean values, numbers (double-precision floating point by default), and strings. Typical data structures such as arrays, sets, lists, and records can be represented using Lua's single native data structure, the table, which is essentially a heterogeneous associative array.

Lua implements a small set of advanced features such as first-class functions, garbage collection, closures, proper tail calls, coercion (automatic conversion between string and number values at run time), coroutines (cooperative multitasking) and dynamic module loading.

By including only a minimum set of data types, Lua attempts to strike a balance between power and size.

Example code [edit]

The classic hello world program can be written as follows:

```
print('Hello World!')
```

It can also be written as

```
io.write('Hello World!\n')
```

or, the example given on the Lua website

```
io.write("Hello world, from ",_VERSION,"!\n")
```

Comments use the following syntax, similar to that of Ada, Eiffel, Haskell, SQL and VHDL:

```
-- A comment in Lua starts with a double-hyphen and runs to the end of the line.

--[[ Multi-line strings & comments
     are adorned with double square brackets. ]]

--[=[ Comments like this can have other --[[comments]] nested. ]=]
```

The factorial function is implemented as a function in this example:

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

Loops   [edit]

Lua has four types of loops: the `while` loop, the `repeat` loop (similar to a `do while` loop), the `for` loop, and the generic `for` loop.

```lua
--condition = true

while condition do
  --statements
end

repeat
  --statements
until condition

for i = first,last,delta do      --delta may be negative, allowing the for loop to count down or up
  --statements
  --example: print(i)
end
```

The generic `for` loop:

```lua
for key, value in pairs(_G) do
  print(key, value)
end
```

would iterate over the table `_G` using the standard iterator function `pairs`, until it returns `nil`.

Functions   [edit]

Lua's treatment of functions as first-class values is shown in the following example, where the print function's behavior is modified:

```lua
do
  local oldprint = print
  -- Store current print function as oldprint
  function print(s)
    --[[ Redefine print function, the usual print function can still be used
         through oldprint. The new one has only one argument.]]
    oldprint(s == "foo" and "bar" or s)
  end
end
```

Any future calls to `print` will now be routed through the new function, and because of Lua's lexical scoping, the old print function will only be accessible by the new, modified print.

Lua also supports closures, as demonstrated below:

```lua
function addto(x)
  -- Return a new function that adds x to the argument
  return function(y)
    --[=[ When we refer to the variable x, which is outside of the current
         scope and whose lifetime would be shorter than that of this anonymous
         function, Lua creates a closure.]=]
    return x + y
  end
end
fourplus = addto(4)
print(fourplus(3))  -- Prints 7

--This can also be achieved by calling the function in the following way:
print(addto(4)(3))
--[[ This is because we are calling the returned function from `addto(4)' with the argument `3' directly.
     This also helps to reduce data cost and up performance if being called iteratively.
]]
```

A new closure for the variable `x` is created every time `addto` is called, so that each new anonymous function returned will always access its own `x` parameter. The closure is managed by Lua's garbage collector, just like any other object.

Tables   [edit]   Lua

Tables are the most important data structures (and, by design, the only built-in composite data type) in Lua, and are the foundation of all user-created types. They are conceptually similar to associative arrays in PHP, dictionaries in Python and Hashes in Ruby or Perl.

A table is a collection of key and data pairs, where the data is referenced by key; in other words, it's a hashed heterogeneous associative array. A key (index) can be any value but nil and NaN. A numeric key of 1 is considered distinct from a string key of `"1"`.

Tables are created using the `{}` constructor syntax:

```
a_table = {} -- Creates a new, empty table
```

Tables are always passed by reference (See Call by sharing):

```
a_table = {x = 10}  -- Creates a new table, with one entry mapping "x" to the number 10.
print(a_table["x"]) -- Prints the value associated with the string key, in this case 10.
b_table = a_table
b_table["x"] = 20   -- The value in the table has been changed to 20.
print(b_table["x"]) -- Prints 20.
print(a_table["x"]) -- Also prints 20, because a_table and b_table both refer to the same table.
```

As record   [edit]

A table is often used as structure (or record) by using strings as keys. Because such use is very common, Lua features a special syntax for accessing such fields. Example:

```
point = { x = 10, y = 20 }   -- Create new table
print(point["x"])            -- Prints 10
print(point.x)               -- Has exactly the same meaning as line above. The easier-to-read
                             --    dot notation is just syntactic sugar.
```

Quoting the Lua 5.1 Reference Manual:[6]

"The syntax var.Name is just syntactic sugar for var['Name'];"

As namespace   [edit]

By using a table to store related functions, it can act as a namespace.

```
Point = {}

Point.new = function(x, y)
  return {x = x, y = y}  --  return {["x"] = x, ["y"] = y}
end

Point.set_x = function(point, x)
  point.x = x  --  point["x"] = x;
end
```

As array   [edit]

By using a numerical key, the table resembles an array data type. Lua arrays are 1-based: the first index is 1 rather than 0 as it is for many other programming languages (though an explicit index of 0 is allowed).

A simple array of strings:

```
array = { "a", "b", "c", "d" }   -- Indices are assigned automatically.
print(array[2])                  -- Prints "b". Automatic indexing in Lua starts at 1.
print(#array)                    -- Prints 4.  # is the length operator for tables and strings.
array[0] = "z"                   -- Zero is a legal index.
print(#array)                    -- Still prints 4, as Lua arrays are 1-based.
```

The length of a table `t` is defined to be any integer index `n` such that `t[n]` is not `nil` and `t[n+1]` is `nil`; moreover, if `t[1]` is `nil`, `n` can be zero. For a regular array, with non-nil values from 1 to a given `n`, its length is exactly that `n`, the index of its last value. If the array has "holes" (that is, nil values between other non-nil values), then `#t` can be any of the indices that directly precedes a `nil` value (that is, it may consider any such nil value as the end of the array).[7]

A two dimensional table:

```
ExampleTable =
{
    {1,2,3,4},
    {5,6,7,8}
}
print(ExampleTable[1][3]) -- Prints "3"
print(ExampleTable[2][4]) -- Prints "8"
```

An array of objects:

```lua
function Point(x, y)        -- "Point" object constructor
  return { x = x, y = y }   -- Creates and returns a new object (table)
end
array = { Point(10, 20), Point(30, 40), Point(50, 60) }   -- Creates array of points
                    -- array = { { x = 10, y = 20 }, { x = 30, y = 40 }, { x = 50, y = 60 } };
print(array[2].y)                                 -- Prints 40
```

Using a hash map to emulate an array normally is slower than using an actual array; however, Lua tables are optimized for use as arrays[8] to help avoid this issue.

Metatables [edit]

Lua

Extensible semantics is a key feature of Lua, and the metatable concept allows Lua's tables to be customized in powerful ways. The following example demonstrates an "infinite" table. For any n, `fibs[n]` will give the nth Fibonacci number using dynamic programming and memoization.

```lua
fibs = { 1, 1 }                              -- Initial values for fibs[1] and fibs[2].
setmetatable(fibs, {
  __index = function(values, n)              --[[ __index is a function predefined by Lua,
                                                 it is called if key "n" does not exist. ]]
    values[n] = values[n - 1] + values[n - 2]  -- Calculate and memorize fibs[n].
    return values[n]
  end
})
```

Object-oriented programming [edit]

Although Lua does not have a built-in concept of classes, they can be implemented using two language features: first-class functions and tables. By placing functions and related data into a table, an object is formed. Inheritance (both single and multiple) can be implemented via the metatable mechanism, telling the object to look up nonexistent methods and fields in parent object(s).

There is no such concept as "class" with these techniques; rather, prototypes are used, as in the programming languages Self or JavaScript. New objects are created either with a factory method (that constructs new objects from scratch), or by cloning an existing object.

Lua provides some syntactic sugar to facilitate object orientation. To declare member functions inside a prototype table, one can use `function table:func(args)`, which is equivalent to `function table.func(self, args)`. Calling class methods also makes use of the colon: `object:func(args)` is equivalent to `object.func(object, args)`.

Creating a basic vector object:

```lua
local Vector = {}
Vector.__index = Vector

function Vector:new(x, y, z)     -- The constructor
  return setmetatable({x = x, y = y, z = z}, Vector)
end

function Vector:magnitude()      -- Another method
  -- Reference the implicit object using self
  return math.sqrt(self.x^2 + self.y^2 + self.z^2)
end

local vec = Vector:new(0, 1, 0) -- Create a vector
print(vec:magnitude())          -- Call a method (output: 1)
print(vec.x)                    -- Access a member variable (output: 0)
```

Internals [edit]

Lua programs are not interpreted directly from the textual Lua file, but are compiled into bytecode which is then run on the Lua virtual machine. The compilation process is typically invisible to the user and is performed during run-time, but it can be done offline in order to increase loading performance or reduce the memory footprint of the host environment by leaving out the compiler.

Like most CPUs, and unlike most virtual machines (which are stack-based), the Lua VM is register-based, and therefore more closely resembles an actual hardware design. The register architecture both avoids excessive copying of values and reduces the total number of instructions per function. The virtual machine of Lua 5 is one of the first register-based pure VMs to have a wide use.[9] Perl's Parrot and Android's Dalvik are two other well-known register-based VMs.

This example is the bytecode listing of the factorial function defined above (as shown by the `luac` 5.1 compiler):[10]

```
function <factorial.lua:1,7> (9 instructions, 36 bytes at 0x8063c60)
1 param, 6 slots, 0 upvalues, 6 locals, 2 constants, 0 functions
        1       [2]     LOADK           1 -1    ; 1
        2       [3]     LOADK           2 -2    ; 2
        3       [3]     MOVE            3 0
        4       [3]     LOADK           4 -1    ; 1
        5       [3]     FORPREP         2 1     ; to 7
        6       [4]     MUL             1 1 5
        7       [3]     FORLOOP         2 -2    ; to 6
        8       [6]     RETURN          1 2
        9       [7]     RETURN          0 1
```

C API [edit]

Lua is intended to be embedded into other applications, and provides a C API for this purpose. The API is divided into two parts: the Lua core and the Lua auxiliary library.[11]

The Lua API's design eliminates the need for manual reference management in C code, unlike Python's API. The API, like the language, is minimalistic. Advanced functionality is provided by the auxiliary library, which consists largely of preprocessor macros which assist with complex table operations.

Stack [edit]

The Lua C API is stack based. Lua provides functions to push and pop most simple C data types (integers, floats, etc.) to and from the stack, as well as functions for manipulating tables through the stack. The Lua stack is somewhat different from a traditional stack; the stack can be indexed directly, for example. Negative indices indicate offsets from the top of the stack. For example, −1 is the top (most recently pushed value), while positive indices indicate offsets from the bottom (oldest value).

Marshalling data between C and Lua functions is also done using the stack. To call a Lua function, arguments are pushed onto the stack, and then the `lua_call` is used to call the actual function. When writing a C function to be directly called from Lua, the arguments are popped from the stack.

Example [edit]

Here is an example of calling a Lua function from C:

```c
#include <stdio.h>
#include <lua.h> //Lua main library (lua_*)
#include <lauxlib.h> //Lua auxiliary library (luaL_*)

int main(void)
{
    //create a Lua state
    lua_State *L = luaL_newstate();

    //load and execute a string
    if (luaL_dostring(L, "function foo (x,y) return x+y end")) {
        lua_close(L);
        return -1;
    }

    //push value of global "foo" (the function defined above)
    //to the stack, followed by integers 5 and 3
    lua_getglobal(L, "foo");
    lua_pushinteger(L, 5);
    lua_pushinteger(L, 3);
    lua_call(L, 2, 1); //call a function with two arguments and one return value
    printf("Result: %d\n", lua_tointeger(L, -1)); //print integer value of item at stack top
    lua_close(L); //close Lua state
    return 0;
}
```

Running this example gives:

```
$ cc -o example example.c -llua
$ ./example
Result: 8
```

Special tables [edit]

The C API also provides some special tables, located at various "pseudo-indices" in the Lua stack. At `LUA_GLOBALSINDEX` prior to Lua 5.2[12] is the globals table, `_G` from within Lua, which is the main namespace. There is also a registry located at `LUA_REGISTRYINDEX` where C programs can store Lua values for later retrieval.

Extension and binding [edit]

It is possible to write extension modules using the Lua API. Extension modules are shared objects which can be used to extend the functionality of the interpreter by providing native facilities to Lua scripts. From the Lua side, such a module appears as a namespace table holding its functions and variables. Lua scripts may load extension modules using `require`,[11] just like modules written in Lua itself.

A growing collection of modules known as *rocks* are available through a package management system called *LuaRocks*,[13] in the spirit of CPAN, RubyGems and Python Eggs. Other modules can be found through the *Lua Addons* directory of the lua-users.org 🔗 wiki.[14]

Prewritten Lua bindings exist for most popular programming languages, including other scripting languages.[15] For C++, there are a number of template-based approaches and some automatic binding generators.

Applications [edit]

Video games [edit]

> Main category: *Lua-scripted video games*

In video game development, Lua is widely used as a scripting language by game programmers, perhaps due to its perceived easiness to embed, fast execution, and short learning curve.[16]

In 2003, a poll conducted by GameDev.net showed Lua as a most popular scripting language for game programming.[17] On 12 January 2012, Lua was announced as a winner of the Front Line Award 2011 from the magazine *Game Developer* in the category Programming Tools.[18]

Other [edit]

> This section **needs additional citations for verification**. Please help improve this article by adding citations to reliable sources. Unsourced material may be challenged and removed. *(February 2012)*

Other applications using Lua include:

- 3DMLW plugin uses Lua scripting for animating 3D and handling different events.
- Adobe Photoshop Lightroom uses Lua for its user interface.
- Aerospike Database uses Lua as its internal scripting language for its 'UDF' (User Defined Function) capabilities - similar to procedures
- Apache HTTP Server can use Lua anywhere in the request process (since version 2.3, via the core mod_lua module).
- Artweaver graphics editor uses Lua for scripting filters.
- Awesome, a window manager, is written partly in Lua, also using it as its configuration file format
- The Canon Hack Development Kit (CHDK), an open source firmware for Canon cameras, uses Lua as one of two scripting languages.
- Celestia, the astronomy educational program, uses Lua as its scripting language.
- Cheat Engine, a memory editor/debugger, enables Lua scripts to be embedded in its "cheat table" files, and even includes a GUI designer.
- Cisco uses Lua to implement Dynamic Access Policies within the Adaptive Security Appliance.
- Conky the Linux system monitoring app uses Lua for advanced graphics.
- Cocos2d uses Lua to build games with their Cocos Code IDE.
- Codea is a Lua editor native to the iOS operating-system.
- Codebymath 🔗 uses the Lua language to teach programming using concepts from basic mathematics.
- Computercraft 🔗 Is a minecraft mod that uses a slightly edited version of Lua. It uses LuaJ to interact with Minecraft itself.
- Custom applications for the Creative Technology Zen X-Fi2 portable media player can be created in Lua.
- Damn Small Linux uses Lua to provide desktop-friendly interfaces for command-line utilities without sacrificing lots of disk space.
- The darktable open-source photography workflow application is scriptable with Lua. [1] 🔗
- Dolphin Computer Access uses Lua scripting to make inaccessible applications accessible for visually impaired computer users with their screen reader – SuperNova.
- Eyeon's Fusion compositor uses embedded Lua and LuaJIT for internal and external scripts and also plugin prototyping.
- A fork of the NES emulator FCE Ultra called FCEUX allows for extensions or modifications to games via Lua scripts.
- Flame, a large and highly sophisticated piece of malware being used for cyber espionage.[19]
- Foldit, a science-oriented game in protein folding, uses Lua for user scripts. Some of those scripts have been the aim of an article in PNAS.[20]
- FreePOPs, an extensible mail proxy, uses Lua to power its web front-end.
- Freeswitch, an open-source telephony platform designed to facilitate the creation of voice and chat driven products in which Lua can be used as a scripting language for call control and call flow among other things.
- Ginga, the middleware for Brazilian Digital Television System (SBTVD or ISDB-T), uses Lua as a script language to its declarative

environment, Ginga-NCL. In Ginga-NCL, Lua is integrated as media objects (called NCLua) inside NCL (Nested Context Language) documents.

- GrafX2, a pixel-art editor, can run Lua scripts for simple picture processing or generative illustration.
- iClone, a 3D real-time animation studio to create animation movies uses Lua in the controls of its new physics simulation.
- The drawing editor Ipe (mainly used for producing figures with LaTeX labeling) uses Lua for its functionality and script extensions.
- Lego Mindstorms NXT and NXT 2.0 can be scripted with Lua using third-party software.[21]
- lighttpd web server uses Lua for hook scripts as well as a modern replacement for the Cache Meta Language.
- Version 2.01 of the profile management software for Logitech's G15 gaming keyboard uses Lua as its scripting language.
- LuaTeX, the designated successor of pdfTeX, allows extensions to be written in Lua.[22]
- MediaWiki [23] uses Lua as a new templating language, which is used on Wikipedia and other wikis.
- The Moonbridge Network Server for Lua Applications in combination with WebMCP, a web development framework, allows complex web applications to be written in Lua (used by LiquidFeedback[24]).
- MySQL Workbench uses Lua for its extensions and add-ons.
- NetBSD has a Lua driver that can create and control Lua states inside the kernel. This allows Lua to be used for packet filtering and creating device drivers. [25][26][27]
- Nginx has a powerful embedded Lua module that provides an API for accessing Nginx facilities like socket handling, for example.[28]
- nmap network security scanner uses Lua as the basis for its scripting language, called *nse*.[29]
- NodeMCU uses Lua in hardware. NodeMCU is an open source hardware platform, which can run Lua directly on ESP8266 Wi-Fi SoC.[30]
- Sierra Wireless AirLink ALEOS GSM / CDMA / LTE gateways allow user applications to be written in Lua.
- The Perimeta session border controller from Metaswitch Networks uses Lua as a scripting language to manipulate SDP data on the fly.[31]
- Ophal is an extensible web development framework mostly written in Lua and Javascript.
- PowerDNS offers extensive Lua scripting for serving and changing DNS answers, fixing up broken servers, and DoS protection.
- Project Dogwaffle Professional offers Lua scripting to make filters through the DogLua filter. Lua filters can be shared between Project Dogwaffle, GIMP, Pixarra Twistedbrush and ArtWeaver.
- Prosody is a cross-platform Jabber/XMPP server written in Lua.
- Reason digital audio workstation, Lua is used to describe Remote codecs.
- Redis is an open source key-value database, in which Lua can be used (starting with version 2.6) to write complex functions that run in the server itself, thus extending its functionality.[32]
- Renoise audio tracker, in which Lua scripting is used to extend functionality.
- Rockbox, the open-source digital audio player firmware, supports plugins written in Lua.
- New versions of SciTE editor can be extended using Lua.
- Screvle, embedded development system with Lua Runtime and on board, web-based Lua Development Environment.
- Snort intrusion detection system includes a Lua interpreter since 3.0 beta release.[33]
- The Squeezebox music players from Logitech support plugins written in Lua on recent models (Controller, Radio and Touch).
- Synalyze It! Pro uses Lua for implementation of custom data types, extended parsing logic and other tasks.
- Torch is an open source deep learning library for Lua.
- Teamspeak has a Lua scripting plugin for modifications.
- Tarantool NoSQL database uses Lua as its stored procedures language.
- TI-Nspire calculators contain applications written in Lua, since TI added Lua scripting support with a calculator-specific API in OS 3+.
- Vim has Lua scripting support starting with version 7.3.[34]
- VLC media player uses Lua to provide scripting support.
- WeeChat IRC client allows scripts to be written in Lua.
- WinGate proxy server allows event processing and policy to execute Lua scripts with access to internal WinGate objects.
- Wireshark network packet analyzer allows protocol dissectors and post-dissector taps to be written in Lua.[35]
- ZeroBrane Studio Lua IDE is written in Lua and uses Lua for its plugins.

References [edit]

1. ^ *a* *b* *c* "About Lua" . Lua.org. Retrieved 2013-06-19.
2. ^ Yuri Takhteyev (21 April 2013). "From Brazil to Wikipedia" . *Foreign Affairs*. Retrieved 25 April 2013.
3. ^ *a* *b* *c* *d* Ierusalimschy, R.; Figueiredo, L. H.; Celes, W. (2007). "The evolution of Lua" (PDF). *Proc. of ACM HOPL III* . pp. 2–1–2–26. doi:10.1145/1238844.1238846 . ISBN 978-1-59593-766-7.
4. ^ "The evolution of an extension language: a history of Lua" . 2001. Retrieved 2008-12-18.
5. ^ Figueiredo, L. H.; Ierusalimschy, R.; Celes, W. (December 1996). "Lua: an Extensible Embedded Language. A few metamechanisms replace a host of features" . *Dr. Dobb's Journal* **21** (12). pp. 26–33.
6. ^ "Lua 5.1 Reference Manual" . 2014. Retrieved 2014-02-27.
7. ^ "Lua 5.1 Reference Manual" . 2012. Retrieved 2012-10-16.

2015/6/6 12:08

8. ^ "Lua 5.1 Source Code". 2006. Retrieved 2011-03-24.
9. ^ Ierusalimschy, R.; Figueiredo, L. H.; Celes, W. (2005). "The implementation of Lua 5.0". *J. Of Universal Comp. Sci.* **11** (7): 1159–1176.
10. ^ Kein-Hong Man (2006). "A No-Frills Introduction to Lua 5.1 VM Instructions" (PDF).
11. ^ *a b* "Lua 5.2 Reference Manual". Lua.org. Retrieved 2012-10-23.
12. ^ "Changes in the API - Lua 5.2 Reference Manual". Lua.org. Retrieved 2014-05-09.
13. ^ "LuaRocks". LuaRocks wiki. Retrieved 2009-05-24.
14. ^ "Lua Addons". Lua-users wiki. Retrieved 2009-05-24.
15. ^ "Binding Code To Lua". Lua-users wiki. Retrieved 2009-05-24.
16. ^ Why is Lua considered a game language? at the Wayback Machine (archived August 20, 2013)
17. ^ Poll Results at the Wayback Machine (archived December 7, 2003)
18. ^ Front Line Award Winners Announced at the Wayback Machine (archived June 15, 2013)
19. ^ Zetter, Kim (28 May 2012). "Meet 'Flame,' The Massive Spy Malware Infiltrating Iranian Computers". Wired News.
20. ^ Algorithm discovery by protein folding game players
21. ^ pbLua Scriptable Operating Systems with Lua
22. ^ "LuaTeX". *luatex.org*. Retrieved 21 April 2015.
23. ^ Technology report, *Wikipedia Signpost* (30 January 2012)
24. ^ LiquidFeedback-Frontend "Public Software Group e. V. - LiquidFeedback Frontend". *public-software-group.org*. Public Software Group. Retrieved 3 April 2015.
25. ^ LUA(4) Man Page netbsd.gw.com. Retrieved on 2015-04-21.
26. ^ NPF Scripting with Lua EuroBSDCon 2014.
27. ^ Scriptable Operating Systems with Lua Dynamic Languages Symposium 2014
28. ^ HttpLuaModule. Wiki.nginx.org. Retrieved on 2013-07-17.
29. ^ "Nmap Scripting Engine". Retrieved 2010-04-10.
30. ^ Huang R. "NodeMCU devkit". *Github*. Retrieved 3 April 2015.
31. ^ "Know Your SBCs" (PDF). Retrieved 2014-03-08.
32. ^ "Redis Lua scripting".
33. ^ "Lua in Snort 3.0". Retrieved 2010-04-10.
34. ^ "Vim documentation: if_lua". Retrieved 2011-08-17.
35. ^ "Lua in Wireshark". Retrieved 2010-04-10.

Further reading [edit]

Books [edit]

- Gutschmidt, T. (2003). *Game Programming with Python, Lua, and Ruby*. Course Technology PTR. ISBN 1-59200-077-0.
- Schuytema, P.; Manyen, M. (2005). *Game Development with Lua*. Charles River Media. ISBN 1-58450-404-8.
- Jung, K.; Brown, A. (2007). *Beginning Lua Programming*. Wrox Press. ISBN 0-470-06917-1.
- Figueiredo, L. H.; Celes, W.; Ierusalimschy, R., eds. (2008). *Lua Programming Gems*. Lua.org. ISBN 978-85-903798-4-3.
- Takhteyev, Yuri (2012). *Coding Places: Software Practice in a South American City*. The MIT Press. ISBN 0262018071. Chapters 6 and 7 are dedicated to Lua, while others look at software in Brazil more broadly.
- Varma, Jayant (2012). *Learn Lua for iOS Game Development*. Apress. ISBN 1430246626.
- Ierusalimschy, R. (2013). *Programming in Lua* (3rd ed.). Lua.org. ISBN 85-903798-5-X. (The 1st ed. is available online.)

Articles [edit]

- Matheson, Ash (29 April 2003). "An Introduction to Lua". GameDev.net. Retrieved 3 January 2013.
- Fieldhouse, Keith (16 February 2006). "Introducing Lua". *ONLamp.com*. O'Reilly Media.
- Streicher, Martin (28 April 2006). "Embeddable scripting with Lua". *developerWorks*. IBM.
- Quigley, Joseph (1 June 2007). "A Look at Lua". *Linux Journal*.
- Hamilton, Naomi (11 September 2008). "The A-Z of Programming Languages: Lua". *Computerworld* (IDG). Interview with Roberto Ierusalimschy.
- Ierusalimschy, Roberto; de Figueiredo, Luiz Henrique; Celes, Waldemar (12 May 2011). "Passing a Language through the Eye of a Needle". *ACM Queue* (ACM). How the embeddability of Lua impacted its design.
- Lua papers and theses

External links [edit]

- Official website
- lua-users.org – Community website for and by users (and authors) of Lua
- eLua – Embedded Lua
- Projects in Lua
- SquiLu Squirrel modified with Lua libraries

Find more about
Lua
at Wikipedia's sister projects

News stories from Wikinews

Textbooks from Wikibooks

Learning resources from Wikiversity