Lua: language for the Web?

@paulcuth
MOONSHINE
A lightweight Lua VM for the browser

[TRY IT] [GET STARTED]

Source available on GitHub.
Starlight
ES6
Generators

Syntax

```javascript
function* gen() {
    yield 1;
    yield 2;
    yield 3;
}

var g = gen(); // "Generator { }"
```
Coroutines

c = coroutine.create(function ()
  coroutine.yield(1)
  coroutine.yield(2)
  coroutine.yield(3)
  end)

print(coroutine.resume(c))
print(coroutine.resume(c))
print(coroutine.resume(c))
Spread operator

Syntax

For function calls:

```javascript
myFunction(...iterableObj);
```

For array literals:

```javascript
[1, 2, 3, ...iterableObj]
```
table.unpack()

-- For function calls:
print(table.unpack(t))

-- For table literals:
{ 1, 2, 3, table.unpack(t) }
Lua is an extension programming language designed to support general procedural programming with data description facilities, also offering good support for object-oriented programming, functional programming, and data-driven programming. Lua is intended to be used as a powerful, light-weight configuration language for any program that needs one. Lua is implemented as a library written in *clean* C (that is, in the common subset of ANSI C and C++).

Being an extension language, Lua has no notion of a "main" program: it only works *embedded* in a host client, called the *embedding program* or simply the *host*. This host program can invoke functions to execute a piece of Lua code, can write and read Lua variables, and can register C functions to be called by Lua code. Through the use of C functions, Lua can be augmented to cope with a wide range of different domains, thus creating customized programming languages sharing a syntax and framework.

The Lua distribution includes a stand-alone embedding program, `lua`, that uses the Lua library to offer a complete Lua interpreter.

Lua is free software, and is provided as usual with no guarantees, as stated in its copyright notice. The implementation described in this manual is available at Lua's official web site, www.lua.org.
1 – Introduction

Lua is an extension programming language designed to support general procedural programming with data description facilities, also offers good support for object-oriented programming, functional programming, and data-driven programming. Lua is intended to be used as a powerful, light-weight configuration language for any program that needs one. Lua is implemented as a library written in clean C (that is, in the common subset of ANSI C and C++).

Being an extension language, Lua has no notion of a "main" program: it only works embedded in a host client, called embedding program or simply the host. This host program can invoke functions to execute a piece of Lua code, can write, read Lua variables, and can register C functions to be called by Lua code. Through the use of C functions, Lua can be augmented to cope with a wide range of different domains, thus creating customized programming languages sharing a syntax framework.

The Lua distribution includes a stand-alone embedding program, lua, that uses the Lua library to offer a complete Lua interpreter.

Lua is free software, and is provided as usual with no guarantees, as stated in its copyright notice. The implementation described in this manual is available at Lua's official web site, www.lua.org.
2003
2003
Rest operator

Syntax

```javascript
function(a, b, ...theArgs) {
  // ...
}
```
Varargs

```plaintext
function(a, b, ...)
  -- ...
end
```
Reference Manual of the Programming Language Lua 2.5

Roberto Jerusalimschy, Luiz Henrique de Figueiredo, Waldemar Celes
lua@tecgraf.puc-rio.br
TeCGraf, Computer Science Department, PUC-Rio

Abstract. Lua is an extension programming language designed to be used as a configuration language for any program that needs one. This document describes version 2.5 of the Lua programming language and the API that allows interaction between Lua programs and their host C programs.

Sumário. Lua é uma linguagem de extensão projetada para ser usada como linguagem de configuração em qualquer programa que precise de uma. Este documento descreve a versão 2.5 da linguagem de programação Lua e a Interface de Programação (API) que permite a interação entre programas Lua e programas C hospedeiros.

1 - Introduction
2 - Environment and Chunks
3 - Types
4 - The Language
   4.1 - Lexical Conventions
   4.2 - Coercion
   4.3 - Adjustment
   4.4 - Statements
   4.5 - Expressions
   4.6 - Function Definitions
   4.7 - Fallbacks
   4.8 - Error Handling
5 - The Application Program Interface
   5.1 - Exchanging Values between C and Lua
   5.2 - Executing Lua Code
   5.3 - Manipulating Lua Objects
   5.4 - Calling Lua Functions
   5.5 - C Functions
   5.6 - References to Lua Objects
6 - Predefined Functions and Libraries
   6.1 - Predefined Functions
   6.2 - String Manipulation
   6.3 - Mathematical Functions
   6.4 - I/O Facilities
7 - The Debugger Interface
   7.1 - Stack and Function Information
   7.2 - Manipulating Local Variables
   7.3 - Hooks
8 - Lua Stand-alone
   Acknowledgments
   Incompatibilities with Previous Versions
   Index

Last update: Thu Jun 26 08:15:42 EST 1997 by lhf.
1996
var handler = {
  get: function(target, name){
    return name in target?
      target[name]:
    37;
  }
};

var p = new Proxy({}, handler);
p.a = 1;
p.b = undefined;

console.log(p.a, p.b); // 1, undefined
console.log('c' in p, p.c); // false, 37
4.7 Fallbacks

Lua provides a powerful mechanism to extend its semantics, called *fallbacks*. Basically, a fallback is a programmer defined function which is called whenever the standard function does not proceed.

Lua supports the following fallbacks, identified by the given strings:

```
``arithmetic
``
called when an arithmetic operation is applied to non numerical operands, or when the binary ^ operation is called. Receives three arguments: the two operands and the operation itself (the operation is unary minus) and one of the following strings describing the offended operator:

```
add sub mul div pow unm
```

Its return value is the final result of the arithmetic operation. The default function issues an error.

```
``order
``
called when an order comparison is applied to non numerical or non string operands. Receives three arguments: the two operands and one of the following strings describing the offending operator:

```
lt gt le ge
```

Its return value is the final result of the comparison operation. The default function issues an error.

```
``concat
``
called when a concatenation is applied to non string operands. Receives the two operands as arguments. Its return value is the final result of the concatenation operation. The default function issues an error.

```
``index
``
called when Lua tries to retrieve the value of an index not present in a table. Receives as arguments the table and the index. Its return value is the final result of the index operation. The default function returns nil.

```
``gettable
``
called when Lua tries to index a non table value. Receives as arguments the non table value and the index. Its return value is the final result of the index operation. The default function issues an error.

```
``settable
``
called when Lua tries to assign an indexed a non table value. Receives as arguments the non table value, the index, and the assigned value. The default function issues an error.

```
``function
``
called when Lua tries to call a non function value. Receives as arguments the non function value and the arguments given in the original call. Its return value is the final result of the call operation. The default function issues an error.

```
``gc
``
called during garbage collection. Receives as argument the table being collected. After each run of the collector this function is called with argument zero. Since during garbage collection, it must be used with great care, and programmers should avoid the creation of new objects (tables or strings) in this function.

```
``error
``
called when an error occurs. Receives as argument a string describing the error. The default function prints the message on the standard error output.
1995

1995

https://upload.wikimedia.org/wikipedia/en/1/13/Toy_Story.jpg
Destructuring assignments

```javascript
1 var a = 1;
2 var b = 3;
3 [a, b] = [b, a];
```
Multiple return values

```javascript
function f() {
    return [1, 2];
}
```
Block scoping

```javascript
if (x > y) {
    let gamma = 12.7 + y;
    i = gamma * x;
}
```
Reference Manual of the Programming Language Lua

Roberto Ierusalimschy
Luiz Henrique de Figueiredo
Waldemar Celes Filho
TeCgraf — PUC-Rio
roberto, lhf, celes@icad.puc-rio.br

May 27, 1994

Abstract

Lua is an embedded programming language designed to be used as a configuration language for any program that needs one. This document describes the Lua programming language and the API that allows interaction between Lua programs and its host C program. It also presents some examples of using the main features of the system.

Sumário

Lua é uma linguagem de extensão projetada para ser usada como linguagem de configuração em qualquer programa que precise de uma. Este documento descreve a linguagem de programação Lua e a Interface de Programação que permite a interação entre programas Lua e o programa C hospedeiro. O documento também apresenta alguns exemplos de uso das principais características do sistema.
1994

https://upload.wikimedia.org/wikipedia/ca/6/68/FIFA_World_Cup_1994_Logo.png
21 years

Justin Bieber, 21
Pop Singer

ES6
Starlight
Hello Web

<!DOCTYPE html>
<html>
  <body>
    <h1>Starlight</h1>
    <script type="application/x-lua">
      print 'Hello Web'
    </script>
    <script src="//cdnjs.cloudflare.com/ajax/libs/babel-core/5.8.34/browser.min.js"></script>
    <script src="../lib/starlight.js" data-run-script-tags></script>
  </body>
</html>
Configuration

```html
<p id="output"></p>

<script>
    window.starlight = {
        config: {
            stdout: {
                writeln: function (message) {
                    document.getElementById('output').innerHTML += message + '<br/>';
                },
            },
            env: {
                getTimeStamp: Date.now.bind(Date)
            }
        }
    };
</script>

<script type="application/x-lua">
    print 'Hello Web'
    print('now: ' .. getTimeStamp())
</script>
```
Modules

```lua
<script type="application/x-lua" data-modname="greeting">
    return {
        greet = function(name)
            print('Hello ' .. name)
        end
    }
</script>

<script type="application/x-lua">
    local greeting = require 'greeting'
greeting.greet 'FOSDEM'
</script>
```
DOMAPI

```lua
<script type="application/x-lua">
local div = window.document:querySelector 'div'

function handleClick ()
    for frame = 0, 7 do
        window:setTimeout(function ()
            div.className = 'frame-'..frame
        end, 50 * frame)
    end
end

window.document:addEventListener('click', handleClick)
</script>
```
window.extract();

local div = document:querySelector 'div'

function handleClick ()
    for frame = 0, 7 do
        setTimeout(function ()
            div.className = 'frame-'..frame
            end, 50 * frame)
    end
end

document:addEventListener('click', handleClick)
</script>
Grunt task

`npm install grunt-starlight --save-dev`
Grunt task

```javascript
grunt.initConfig({
    starlight: {
        trails: {
            src: 'src/script/trails.lua',
            dest: 'dist/script/trails.es6.js',
        }
    },
    browserify: {
        options: {
            transform: ['babelify'],
        },
        trails: {
            files: {
                'dist/script/trails.js': ['dist/script/trails.es6.js'],
            },
        },
    },
});
```
Roadmap

• Source mapping
• Gulp task
• `<script type="application/x-lua" src="…">`
• Plug-in system (coroutines)
Please help

https://github.com/paulcuth/starlight

@paulcuth
Questions?

https://github.com/paulcuth/starlight

@paulcuth