Themenabend: Lua

twobit

c3d2

23. November 2011

Outline

Outline

Position Nov 2011	Position Nov 2010	Delta in Position	Programming Language	Ratings Nov 2011	Delta Nov 2010	Status
1	1	=	Java	17.874%	-0.63%	A
2	2	=	C	17.322%	+0.61%	A
3	3	=	C++	8.084%	-1.41%	A
4	5	↑	C#	7.319%	+1.61%	A
5	4	↓	PHP	6.096%	-1.72%	A
6	8	↑↑	Objective-C	5.983%	+2.79%	A
7	7	=	(Visual) Basic	5.041%	-0.43%	A
8	6	↓↓	Python	3.617%	-2.06%	A
9	11	↑↑	JavaScript	2.565%	+0.90%	A
10	9	↓	Perl	2.078%	-0.39%	A
11	10	↓	Ruby	1.502%	-0.40%	A
12	20	↑↑↑↑↑↑↑↑	PL/SQL	1.438%	+0.78%	A
13	13	=	Lisp	1.182%	+0.09%	A
14	15	↑	Pascal	0.991%	+0.21%	A
15	21	↑↑↑↑↑↑	MATLAB	0.955%	+0.32%	A--
16	12	↓↓↓↓	Delphi/Object Pascal	0.872%	-0.77%	A
17	23	↑↑↑↑↑↑	ABAP	0.847%	+0.25%	A--
18	22	↑↑↑↑	Lua	0.635%	+0.02%	A-
19	16	↓↓↓	Ada	0.622%	-0.07%	B
20	19	↓	RPG (OS/400)	0.620%	-0.04%	B

Design-Ziele

- Simplicity

Design-Ziele

- Simplicity
- Size

Design-Ziele

- Simplicity
- Size
- Performance

Design-Ziele

- Simplicity
- Size
- Performance
- Portability

Design-Ziele

- Simplicity
- Size
- Performance
- Portability
- Embeddability

Wer hat's erfunden?

- Roberto Ierusalimschy

Wer hat's erfunden?

- Roberto Ierusalimschy
- Waldemar Celes

Wer hat's erfunden?

- Roberto Ierusalimschy
- Waldemar Celes
- Luiz Henrique de Figueiredo


the programming language
Lua

Features

- first-class functions

Features

- first-class functions
- lexical scoping

Features

- first-class functions
- lexical scoping
- proper tail call

Features

- first-class functions
- lexical scoping
- proper tail call
- coroutines

Features

- first-class functions
- lexical scoping
- proper tail call
- coroutines
- monkey patching

Features

- first-class functions
- lexical scoping
- proper tail call
- coroutines
- monkey patching
- open source

Portability

- Cameras (Canon)

Portability

- Cameras (Canon)
- Keyboards (Logitech)

Portability

- Cameras (Canon)
- Keyboards (Logitech)
- Drucker (Olivetty & Océ)

Portability

- Cameras (Canon)
- Keyboards (Logitech)
- Drucker (Olivetty & Océ)
- PSP, PS3, OpenPandora, Nintendo DS

Portability

- Cameras (Canon)
- Keyboards (Logitech)
- Drucker (Olivetty & Océ)
- PSP, PS3, OpenPandora, Nintendo DS
- iPhone, Android

Projekte

- awesome

Projekte

- awesome
- Grim Fandango

Projekte

- awesome
- Grim Fandango
- Codea

Projekte

- awesome
- Grim Fandango
- Codea
- löve (http://stabyourself.net/)

Projekte

- awesome
- Grim Fandango
- Codea
- löve (http://stabyourself.net/)
- Lego Mindstorms NXT

Projekte

- awesome
- Grim Fandango
- Codea
- löve (http://stabyourself.net/)
- Lego Mindstorms NXT
- WoW

Projekte

- awesome
- Grim Fandango
- Codea
- löve (http://stabyourself.net/)
- Lego Mindstorms NXT
- WoW
- renoise

Projekte

- awesome
- Grim Fandango
- Codea
- löve (http://stabyourself.net/)
- Lego Mindstorms NXT
- WoW
- renoise
- vim, vlc, Wireshark, nmap

Projekte

- awesome
- Grim Fandango
- Codea
- löve (http://stabyourself.net/)
- Lego Mindstorms NXT
- WoW
- renoise
- vim, vlc, Wireshark, nmap
- luvit

Projekte

- awesome
- Grim Fandango
- Codea
- löve (http://stabyourself.net/)
- Lego Mindstorms NXT
- WoW
- renoise
- vim, vlc, Wireshark, nmap
- luvit
- . . .

Poll Results		
Which language do you use for scripting in your game engine?		
Python (pure)	6.98%	48
Python (stackless)	0%	0
Ruby	1.16%	8
Perl	1.31%	9
C (with co-routines)	9.75%	67
Lisp	1.45%	10
TCL	0.582%	4
Lua	20.5%	141
I made my own	26.3%	181
My engine doesn't have scripting	27.3%	188
Other	4.51%	31
[Previous Polls]	**Total Votes: 687**	

Outline


the programming language
Lua

Code: "hello.lua"

```
1  #!/usr/bin/lua
2
3  -- say hello
4  -- try it out: http://www.lua.org/demo.html
5
6  print("Hello, world!")
```

Datentype

- nil

Datentype

- nil
- bool

Datentype

- nil
- bool
- number

Datentype

- nil
- bool
- number
- string

Datentype

- nil
- bool
- number
- string
- function

Datentype

- nil
- bool
- number
- string
- function
- coroutine

Datentype

- nil
- bool
- number
- string
- function
- coroutine
- table

Datentype

- nil
- bool
- number
- string
- function
- coroutine
- table
- userdata

Code: "scoping.lua"

```lua
do
    local x = 1
    do
        local x = 3
        print(x) -- 3
        y = "global"
    end
    print(x) -- 1
end
print(x) -- nil
print(y) -- global
```

Code: "control1.lua"

```lua
-- ite and numeric for
for i = 1, 100 do
    if i % 3 == 0 and i % 5 == 0 then
        print("FizzBuzz")
    elseif i % 3 == 0 then
        print("Fizz")
    elseif i % 5 == 0 then
        print("Buzz")
    else
        print(i)
    end
end
```

Code: "control2.lua"

```lua
-- while
local x = 1
while x < 100 do
    x = x * 2
end
print(x)

-- repeat until
repeat
    local input = io.read()
    print(input)
until input == "exit"
```

Tables

- assoziatives Array

Tables

- assoziatives Array
- jede Value kann Index sein

Tables

- assoziatives Array
- jede Value kann Index sein
- effiziente Implementierung

Tables

- assoziatives Array
- jede Value kann Index sein
- effiziente Implementierung
- DIY Objekt-System

Tables

- assoziatives Array
- jede Value kann Index sein
- effiziente Implementierung
- DIY Objekt-System
  - Liste

Tables

- assoziatives Array
- jede Value kann Index sein
- effiziente Implementierung
- DIY Objekt-System
  - Liste
  - Hashtabelle

Tables

- assoziatives Array
- jede Value kann Index sein
- effiziente Implementierung
- DIY Objekt-System
  - Liste
  - Hashtabelle
  - Set

Tables

- assoziatives Array
- jede Value kann Index sein
- effiziente Implementierung
- DIY Objekt-System
  - Liste
  - Hashtabelle
  - Set
  - Module

Tables

- assoziatives Array
- jede Value kann Index sein
- effiziente Implementierung
- DIY Objekt-System
  - Liste
  - Hashtabelle
  - Set
  - Module
  - Objekt

Tables

- assoziatives Array
- jede Value kann Index sein
- effiziente Implementierung
- DIY Objekt-System
  - Liste
  - Hashtabelle
  - Set
  - Module
  - Objekt
  - ...

Code: "list.lua"

```lua
local list = { "one", "two", "three" }
list[4] = "four"

for i = 1, #list do -- numeric for
    print(i, list[i])
end

for i, v in ipairs(list) do -- generic for
    print(i, v)
end

print(list[5]) -- ???
```

Code: "hash.lua"

```lua
local hash = {
   ["null"] = 0,
   ["eins"] = 1,
   ["zwei"] = 2
}
hash["drei"] = 3

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

Code: "set.lua"

```
1  local set = {}
2
3  local x = 0xc3d2
4  local y = 0xcccb
5
6  set[x] = true
7  set[y] = true
8
9  if set[50130] then
10     print("found it!")
11 end
```

Code: "module.lua"

```lua
print("Enter your name, please.")

local name = io.read()

print(string.format("Hello %s, nice to meet you.", name))

print("Enter a number, please.")

local n = io.read()
local s = math.sin(n)

print("Sinus is " .. s)
```

Code: "object.lua"

```lua
local player = { x = 0 }

function player.move(self, dx)
    self.x = self.x + dx
end
player.move(player, 3)

 -- syntactic sugar
function player:move(dx)
    self.x = self.x + dx
end
player:move(3)
```

Metatables

- jeder Table kann einen haben

Metatables

- jeder Table kann einen haben
- spezielle Schlüssel

Metatables

- jeder Table kann einen haben
- spezielle Schlüssel
  - __call

Metatables

- jeder Table kann einen haben
- spezielle Schlüssel
  - __call
  - __add

Metatables

- jeder Table kann einen haben
- spezielle Schlüssel
  - __call
  - __add
  - __mul

Metatables

- jeder Table kann einen haben
- spezielle Schlüssel
  - __call
  - __add
  - __mul
  - __concat

Metatables

- jeder Table kann einen haben
- spezielle Schlüssel
  - __call
  - __add
  - __mul
  - __concat
  - __mod

Metatables

- jeder Table kann einen haben
- spezielle Schlüssel
  - __call
  - __add
  - __mul
  - __concat
  - __mod
  - __eq

Metatables

- jeder Table kann einen haben
- spezielle Schlüssel
  - __call
  - __add
  - __mul
  - __concat
  - __mod
  - __eq
  - __index

Metatables

- jeder Table kann einen haben
- spezielle Schlüssel
  - __call
  - __add
  - __mul
  - __concat
  - __mod
  - __eq
  - __index
  - __newindex

Metatables

- jeder Table kann einen haben
- spezielle Schlüssel
  - __call
  - __add
  - __mul
  - __concat
  - __mod
  - __eq
  - __index
  - __newindex
  - ...

Code: "meta1.lua"

```lua
local pet = {}
pet.noise = "squeak"

function pet:greet()
    print(self.noise)
end

pet:greet() -- "squeak"
```

Code: "meta2.lua"

```lua
local cat = { noise = "miaow" }

-- make pet prototype of cat
setmetatable(cat, { __index = pet })

cat:greet() -- "miaow"
pet:greet() -- "squeak"
```

Code: "class.lua"

```lua
Class = {}
function Class:new(o)
    o = o or {}
    setmetatable(o, self)
    self.__index = self
    self.__call = getmetatable(self).__call
    return o
end
setmetatable(Class, { __call = function(self, ...)
    local o = self:new()
    if o.init then o:init(...) end
    return o
end })
```

Code: "game1.lua"

```lua
require "class"

Enemy = Class:new { strength = 5 }
function Enemy:init(x, y)
    self.x = x
    self.y = y
end

function Enemy:attack()
    -- ...
end
```

Code: "game2.lua"

```lua
Boss = Enemy:new { strength = 100 }

function Boss:attack()
    -- some other implementation
end

local enemylist = {
    Enemy(-10, 10), -- calls init
    Enemy(  0, 13),
    Enemy( 10, 10),
    Boss(0, 5)
}
```

Code: "privacy.lua"

```lua
-- closures
function new_player()
    local x = 0
    local function move(dx)
        x = x + dx
        print(x)
    end
    return { move = move }
end

local player = new_player()
player.move(10) -- no colon here!!
```

Code: "nicestring.lua"

```lua
-- python-like string formating
getmetatable("").__mod = function(s, a)
    if not a then
        return s
    elseif type(a) == "table" then
        return s:format(unpack(a))
    else
        return s:format(a)
    end
end

print("0x%02X" % 42) -- 0x2A
```

Ende

- Mechanismen

Ende

- Mechanismen
- Flexibilität

Ende

- Mechanismen
- Flexibilität
- keine vorgeschriebene Herangehensweise

Ende

- Mechanismen
- Flexibilität
- keine vorgeschriebene Herangehensweise
- no batteries

Fragen

- Traut Euch!