

# Introduktion til Lua

Thomas B. Rasmussen

FLUG - Fyns Linux User Group

25. Oktober, 2012



# Hvad er Lua?

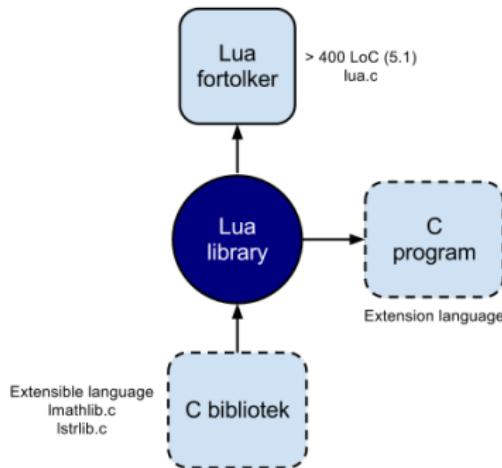
- Lua er et kraftfuldt, letvægts, embeddable scripting sprog
- Udgivet under MIT licens
- Lua er udviklet i standart C (for portabilitet)
- Populært som embedded sprog i fx. C og C++ programmer
- Udviklet i Brasilien
- Lua er portugisisk for måne



# Lua releases

- 5.2 – 16. december 2011
- 5.1 – 21. februar 2006
- 5.0 – 11. april 2003
- 4.0 – 6. november 2000
- 3.0 – 1. juli 1997
- 2.1 – 7. februar 1995
- 1.0 – 28. juli 1993

# Lua oversigt



Lua 5.2 source code <http://www.lua.org/source/5.2/>

# Hvem og hvor benyttes Lua?

- Adobe Lightroom
- World of Warcraft
- Lighttpd
- Wikipedia – Lua som template sprog
- Redis 2.6.0+
- Xavante – webserver skrevet i Lua
- PL/Lua – loadable language for PostgreSQL
- Samsung – Samsung Game Framework (Lua og SDL)
- Ginga – brasiliansk Digital TV Middleware System



- Installer igennem de fleste pakkesystemer  
`sudo apt-get install lua`
- Hent \*.tar.gz fra <http://www.lua.org/download.html>
- Lua for Windows batteries included  
<http://code.google.com/p/luaforwindows/>



- Lua script

```
$lua hello-world.lua  
$hello world
```

- Lua interaktiv

```
$lua  
Lua 5.1.4 Copyright (C) 1994-2008 Lua.org,  
PUC-Rio  
> print "Hello world"  
Hello world  
>^C
```



```
1 -- hello-world.lua
2
3 -- Dette er en kommentar
4
5 --[[[
6   Dette er en kommentar
7   der fylder flere linier...
8 ]]--
9
10 print "Hello world\n"
11 print("Hello world\n")
```

## Lua keywords

and	break	do	else
elseif	end	false	for
function	goto <sup>1</sup>	if	in
local	nil	not	or
repeat	return	then	true
until	while		

<http://www.lua.org/manual/5.2/manual.html#3.1>

<sup>1</sup>Først med i Lua fra 5.2

- Nil – svarende til NULL i mange andre sprog
- Boolean – true, false
- Number – heltal, kommatal...
- String
- Table
- Function
- Userdata & Threads

```
1 -- typer.lua
2
3 print(type(nil))
4 -- nil
5 print(type(false))
6 -- boolean
7 print(type("Hello world\n"))
8 -- string
9 print(type(42+3.0+5e2))
10 --number
11 print(type(print))
12 -- function
13 print(type({}))
14 -- table
```

# Typer

## Table

- Tabeller implementere associative arrays
- Det er værd at bemærke at Lua er 1 indekseret...
- Tabeller er mere end bare arrays...



```
1 -- table.lua
2
3 t           = {'test', 'hest', x='fest'}
4 t[4]        = "hello kitty"
5 t['first']  = 'hello world'
6 print(t[0])    -- nil
7 print(t[1])    -- test
8 print(#t)      -- 4
9 print(t.first) -- hello world
10 print(t.x)     -- fest
11
12 for i = 1, #t do
13     print(t[i])
14 end
15 -- test
16 -- hest
17 -- nil
18 -- hello kitty
19
20 for k,v in pairs(t) do
21     print(k, v)
22 end
23 -- 1      test
24 -- 2      hest
25 -- 4      hello kitty
26 -- first  hello world
27 -- x      fest
```

```
1 -- matrice.lua
2
3 m = {}
4 m[1] = {1,2,3,4,5}
5 m[2] = {2,3,4,5,6}
6 m[3] = {3,4,5,6,7}
7 m[4] = {4,5,6,7,8}
8 m[5] = {5,6,7,8,9}
9
10 for y = 1, #m do
11     for x = 1, #m[y] do
12         io.write(" " .. m[x][y])
13     end
14     io.write("\n")
15 end
16
17 -- 1 2 3 4 5
18 -- 2 3 4 5 6
19 -- 3 4 5 6 7
20 -- 4 5 6 7 8
21 -- 5 6 7 8 9
```

Funktioner i Lua er first-class values, dvs

- de kan gemmes i variabler
- sendes med som argumenter til funktioner
- og returneres fra funktioner

```
1 -- functions.lua
2
3 function f(a)
4     print(a)
5 end
6
7 f('test')      -- test
8 x = f
9 x('tester')   -- tester
10
11 function f1(x)
12     return function(y)
13         return x + y
14     end
15 end
16
17 fourplus = f1(4)
18 print(fourplus(8))    -- 12
19 print(fourplus(10))   -- 14
20
21 function f2(func)
22     func()
23 end
24
25 f2(function() print('der') end) -- der
```

# Expressions

## Aritmetiske operatorer

- + – addition
- - – substraktion
- \* – multiplikation
- / – division
- ^ – eksponential
- % – modulo<sup>2</sup>
- Lua har ikke operatorerne ++, +=, \*=, -= etc.

---

<sup>2</sup>Først fra Lua 5.1

# Expressions

## Relationelle operatorer

- `<-` mindre end
- `>-` større end
- `<=` – mindre end lig med
- `>=` – større end lig med
- `==` – lig med
- `~=` – forskellig fra

# Expression

## Logiske operatorer

- and
- or
- not
- Lua har ikke bitwise operatorer som & og |

```
1 -- logical-operators.lua
2
3 if true and false then
4     print('w00t')
5 end
6
7 if false or true then
8     print("Enten eller")
9 end
10 -- Enten eller
11
12 if not false then
13     print("Ikke falsk")
14 end
15 -- Ikke falsk
```

# Statements

## Assignment

- = – assignment

```
1 -- assignment.lua
2
3 a = 'aA'
4 print(a)
5 -- aA
6
7 x, y, z = 1, 2, 3
8 print(x, y, z)
9 -- 1 2 3
10
11 x, y, z = 1
12 print(x, y, z)
13 -- 1 nil nil
```

# Statements

## Lokale variabler

- Variabler er som standart globale
- Lokale variabler skal erklæres som local<sup>3</sup>
- Udover at man kan spare sig selv for en del fejl kan der være en performance gevindst ved at erklære variabler lokale

---

<sup>3</sup>Som Javascripts var

```
1 -- local.lua
2
3 function foo()
4     a = "test"
5     local b = "hest"
6 end
7
8 print(a, b)
9 -- nil nil
10
11 foo()
12
13 print(a, b)
14 -- test nil
```

# Statement

## Kontrol strukturer

- if then else
- while
- repeat
- for
- break og return

```
1 -- control-structures-if.lua
2
3 if true ~= false and 10 < 11 then
4     print("If")
5 else
6     print("Else")
7 end
8 -- If
```

```
1 -- control-structures.lua
2
3 for i = 1, 3 do
4     print(i)
5 end
6 -- 1
7 -- 2
8 -- 3
9
10 for i = 1, 5, 2 do
11     print(i)
12 end
13 -- 1
14 -- 3
15 -- 5
```

```
1 -- control-structures-for-in.lua
2
3 days = {'Mandag', 'Tirsdag', 'Onsdag', 'Torsdag', 'Fredag'}
4 for idx, day in pairs(days) do
5     print(idx, day)
6 end
7 -- 1      Mandag
8 -- 2      Tirsdag
9 -- 3      Onsdag
10 -- 4     Torsdag
11 -- 5     fredag
```

```
1 -- control-structures-while-repeat.lua
2
3 local x = 1
4 while x <= 3 do
5     print(x)
6     x = x + 1
7 end
8 -- 1
9 -- 2
10 -- 3
11
12 repeat
13     x = x + 1
14     print(x)
15 until x == 6
16 -- 4
17 -- 5
18 -- 6
```

# Functions

- Lua funktioner understøtter multiple return values
- Variabelt antal argumenter ...
- Named arguments
- Closures

```
1 -- functions-advanced.lua
2
3 function foo(bar)
4     return bar*2, bar/2
5 end
6
7 a, b = foo(4)
8 print(a, b)
9 -- 8    2
10 _, b = foo(8)
11 print(b)
12 -- 4
13
14 obj = {}
15 obj.func = function() print "Hello world\n" end
16 obj.func()
17 -- Hello world
```

```
1 -- functions-advanced-varargs.lua
2
3 function varArgs(...)
4     for k, v in ipairs{...} do
5         print(k, v)
6     end
7 end
8
9 varArgs('foo', 'bar', 42)
10 -- 1    foo
11 -- 2    bar
12 -- 3    42
```

```
1 -- functions-advanced-named.lua
2
3 function Window(arg)
4     print(arg.x)
5     print(arg.y)
6     print(arg.width)
7     print(arg.height)
8     print(arg.title)
9 end
10
11 w = Window({width=640, height=480, title='My window', x=0, y=0})
12 w = Window{width=640, height=480, title='My window', x=0, y=0}
13 -- 0
14 -- 0
15 -- 640
16 -- 480
17 -- My window
```

```
1 -- functions-advanced-closure.lua
2
3 function counter()
4     local i = 0
5     return function()
6         i = i + 1
7         return i
8     end
9 end
10
11 c1 = counter()
12 print(c1()) -- 1
13 print(c1()) -- 2
14 c2 = counter()
15 print(c2()) -- 1
16 print(c1()) -- 3
17 print(c2()) -- 2
```

## Minder om threads

- egen stak
- egne lokale variabler
- egen instruktions pointer
- deler globale variabler

Og så alligevel ikke

- er ikke concurrent, der kører aldrig mere end en af gangen
- non-preemptive multithreading. Kan ikke stoppes udefra når de kører, suspenderer selv
- forsimpler da synkronisering derved ikke er et problem



# Coroutines

## Tilstande

- suspended (pause, sleep)
- running
- dead
- normal (coroutine resumer anden coroutine)



# Coroutines

## Funktioner

Modulet coroutine indeholder de følgende funktioner

- `create(...)`
- `status(...)`
- `resume(...)`
- `yield(...)`



```
1 -- coroutine.lua
2
3 function produce()
4     for i = 1, 2 do
5         print(coroutine.status(co))
6         coroutine.yield(i)
7     end
8     return 42
9 end
10
11 co = coroutine.create(produce)
12 print(type(co)) -- thread
13
14 print(coroutine.status(co)) -- suspended
15 print(coroutine.resume(co)) -- running
16             -- true      1
17 print(coroutine.resume(co)) -- running
18             -- true      2
19 print(coroutine.status(co)) -- suspended
20 print(coroutine.resume(co)) -- true      42
21 print(coroutine.resume(co)) -- false cannot resume dead coroutine
22 print(coroutine.status(co)) -- dead
```

# Metatables & Metamethods

Metatables og metametoder kan benyttes til at

- udvide Lua's indbyggede funktionalitet og semantik
- ændre eksisterende metoder og operatorer
- setmetatable()
- getmetatable()



# Metatables & Metamethods

## Aritmetiske metametoder

- `__add` – `+`
- `__sub` – `-`
- `__mul` – `*`
- `__div` – `/`
- `__unm` – `-`
- `__mod` – `%`
- `__pow` – `^`
- `__concat` – `..`

# Metatables & Metamethods

## Relationelle metametoder

- \_\_lt – <
- \_\_le – <=
- \_\_eq – =

Der findes ikke tilsvarende metametoder for ~=, > og >= da Lua oversætter vha. af de ovenstående og not operatoren.



# Metatables & Metamethods

## Library metametoder

- \_\_tostring – tostring()
- \_\_metatable – setmetatable()
- \_\_len – #<sup>4</sup>

---

<sup>4</sup>Først fra Lua 5.2



```
1 -- meta.lua
2
3 t = {a = 'A', b='B'}
4 mt = {}
5 setmetatable(t, mt)
6
7 print(t)
8 -- table: 0x1db0970
9
10 mt.__tostring = function (t)
11     return 'a = ' .. t.a .. ", b = " .. t.b
12 end
13
14 print(t)
15 -- a = A, b = B
```

# Metatables & Metamethods

## Table-Access metametoder

- `__index` – når et ikke eksisterende felt læses fra tabel rammes denne metode
- `__newindex` – ved skrivning til ikke eksisterende felt rammes denne metode
- `rawget(tbl, k)` – tilgår felt uden om `__index`
- `rawset(tbl, k, v)` – opdaterer felt uden om `__newindex`



```
1 -- table-access-get.lua
2
3 t = {a = 'aaa', b = 'bbb'}
4 mt = {}
5 setmetatable(t, mt)
6 mt.__index = function () return 10 end
7 print(t.a)
8 -- aaa
9 print(t.b)
10 -- bbb
11 print(t.c)
12 -- 10
13 print(rawget(t, 'a'))
14 -- aaa
15 print(rawget(t, 'c'))
16 -- nil
```

```
1 -- table-access-set.lua
2
3 t = {a='aaa', b='bbb'}
4 mt = {}
5 setmetatable(t, mt)
6
7 t.a = 'AAA'
8 t.c = 'CCC'
9
10 for k,v in pairs(t) do
11     print(k, v)
12 end
13 -- a    AAA
14 -- c    CCC
15 -- b    bbb
16
17 mt.__newindex = function(tbl, k, v) return end
18
19 t.b = 'BBB'
20 t.d = 'DDD'
21
22 for k,v in pairs(t) do
23     print(k, v)
24 end
25 -- a    AAA
26 -- c    CCC
27 -- b    BBB
```

- `_G` – er en tabel der indeholder globale værdier

Der er ændret en del i 5.2. Bl.a. er funktionerne `setfenv()` og `getfenv()` er deprecated

```
1 -- environment.lua
2
3 a      = 'Var_A'
4 local b = 'Var_B'
5 _G['c'] = 'Var_C'    -- c = Var_C
6
7 function foo(bar)
8     print(bar)
9 end
10
11 print(a, b, c)
12
13 print("====")
14 print("= Environment =")
15 print("====")
16
17 for k, v in pairs(_G) do
18     print(k, v)
19 end
20
21 print("====")
22
23 -- Overskriv global funktion
24 _G['print'] = function(str) io.write(str:reverse() .. "\n") end
25 print("Hello world")
```

```
Var_A  Var_B  Var_C
=====
= Environment =
=====
a      Var_A
string  table: 0x1c48340
xpcall  function: 0x1c479c0
package table: 0x1c48b50
 tostring     function: 0x1c478a0
print    function: 0x1c47a40
os       table: 0x1c4b2f0
unpack   function: 0x1c47960
require  function: 0x1c49550
getfenv  function: 0x1c47430
setmetatable function: 0x1c477e0
next     function: 0x1c47600
assert   function: 0x1c47310
tonumber  function: 0x1c47840
io       table: 0x1c4a910
rawequal  function: 0x1c47aa0
collectgarbage function: 0x1c47370
arg      table: 0x1c4e060
getmetatable function: 0x1c476c0
module   function: 0x1c494f0
rawset   function: 0x1c47b60
foo      function: 0x1c4f680
c       Var_C
math     table: 0x1c4c720
debug    table: 0x1c4d7f0
pcall   function: 0x1c47660
table   table: 0x1c487a0
newproxy  function: 0x1c48560
type    function: 0x1c47900
coroutine table: 0x1c48600
_G      table: 0x1c466b0
select   function: 0x1c46700
gcinfo   function: 0x1c473d0
pairs    function: 0x1c470b0
rawget   function: 0x1c47b00
loadstring function: 0x1c475a0
ipairs   function: 0x1c47010
_VERSION  Lua 5.1
dofile   function: 0x1c47480
setfenv  function: 0x1c47780
load     function: 0x1c47540
error    function: 0x1c474e0
loadfile  function: 0x1c47720
=====
dlrow olleH
```

# Moduler & Packages

Moduler indlæses med require

```
require "module"  
module.foo()
```

Man kan omdøbe moduler og deres funktioner så man slipper for at skrive så meget

```
local m = require "module"  
m.foo()  
local f = m.foo  
f()
```

require kigger efter module.lua i \$LUA\_PATH



```
1 -- module.lua
2
3 local M = {}
4 complex = M      -- modul navn
5
6 M.i = {r=0, i=1}
7
8 function M.new(r, i) return {r=r, i=i} end
9
10 function M.add(c1, c2)
11     return M.new(c1.r + c2.r, c1.i + c2.i)
12 end
```

```
1 -- module-usage.lua
2
3 require "module"
4
5 a = complex.new(1,5)
6 b = complex.new(2,8)
7 c = complex.add(a, b)
8
9 print(c.r, c.i)
10 -- 3      13
```

# Moduler & Packages

## Kompilerede moduler

I Lua er det muligt at prækompilere sin kode til bytecode

```
$luac -o module.out module.lua
```

- hurtigere load af programmet (samme afviklings tid)
- mulighed for binær distribution
- fjerne bytecode compileren (mindre memory footprint)



```
1 -- module-usage-luac.lua
2
3 -- Ved kompiledede moduler benyttes 'dofile'
4 -- istedet for 'require'
5 dofile "module.out"
6 -- alternativt:
7 -- m = loadfile "module.out"
8 -- m()
9
10 a = complex.new(1,5)
11 b = complex.new(2,8)
12 c = complex.add(a, b)
13
14 print(c.r, c.i)
15 -- 3      13
```

Det er muligt at få listet bytecode i det binære modul

```
$luac -l module.out
```

Derudover kan man benytte luac til at checke for syntax fejl

```
$luac -p module.lua
```

```

main <module.lua:0,0> (12 instructions, 48 bytes at 0x11de530)
0+ params, 2 slots, 0 upvalues, 1 local, 7 constants, 2 functions
 1 [1] NEWTABLE 0 0 0
 2 [2] SETGLOBAL 0 -1 ; complex
 3 [4] NEWTABLE 1 0 2
 4 [4] SETTABLE 1 -3 -4 ; "r" 0
 5 [4] SETTABLE 1 -2 -5 ; "i" 1
 6 [4] SETTABLE 0 -2 1 ; "i" -
 7 [6] CLOSURE 1 0 ; 0x11de7c0
 8 [6] SETTABLE 0 -6 1 ; "new" -
 9 [10] CLOSURE 1 1 ; 0x11de8e0
10 [10] MOVE 0 0
11 [8] SETTABLE 0 -7 1 ; "add" -
12 [10] RETURN 0 1

function <module.lua:6,6> (5 instructions, 20 bytes at 0x11de7c0)
2 params, 3 slots, 0 upvalues, 2 locals, 2 constants, 0 functions
 1 [6] NEWTABLE 2 0 2
 2 [6] SETTABLE 2 -1 0 ; "r" -
 3 [6] SETTABLE 2 -2 1 ; "i" -
 4 [6] RETURN 2 2
 5 [6] RETURN 0 1

function <module.lua:8,10> (11 instructions, 44 bytes at 0x11de8e0)
2 params, 6 slots, 1 upvalue, 2 locals, 3 constants, 0 functions
 1 [9] GETUPVAL 2 0 ; M
 2 [9] GETTABLE 2 2 -1 ; "new"
 3 [9] GETTABLE 3 0 -2 ; "r"
 4 [9] GETTABLE 4 1 -2 ; "r"
 5 [9] ADD 3 3 4
 6 [9] GETTABLE 4 0 -3 ; "i"
 7 [9] GETTABLE 5 1 -3 ; "i"
 8 [9] ADD 4 4 5
 9 [9] TAILCALL 2 3 0
10 [9] RETURN 2 0
11 [10] RETURN 0 1

```

- `$LUA_PATH`<sup>5</sup> – system variabel der fortæller Lua hvor den skal lede efter moduler
- `package.path` – Lua variabel der inderholder `$LUA_PATH`
- `$LUA_CPATH` – fortæller Lua hvor den skal lede efter C moduler
- `package.cpath` – Lua variabel der inderholder `$LUA_CPATH`

```
$cat ~/.bashrc | grep 'LUA_PATH'  
export LUA_PATH='~/lua/LIBS/?..lua;  
/usr/local/share/lua/5.1/?..lua;;'
```

---

<sup>5</sup>Samme som Java `$CLASSPATH`

```
1 -- path.lua
2
3 print(package.path)
4 -- /home/thor/LIBS/??.lua;/usr/local/share/lua/5.1/??.lua; \
5 -- ./??.lua;/usr/local/share/lua/5.1/??.lua;
6 print(package.cpath)
7 -- ./??.so;/usr/local/lib/lua/5.1/??.so;/usr/lib/lua/??.so; \
8 -- /usr/local/lib/lua/5.1/loadall.so
```

# Object-Oriented Programming

Selv om Lua ikke har class begrebet er det muligt at implementere de OOP begreber vi kender fra andre sprog

- Objekter
- Konstruktører
- Nedarvning
- Polymorphi
- Private/Public

Til OOP udnytter vi Lua's tabel datatype og at funktioner er first-class values



```
1 -- oop.lua
2
3 Account = { balance = 0 }
4
5 function Account.deposit(x)
6     Account.balance = Account.balance + x
7 end
8
9 a = Account
10 b = Account
11
12 a.deposit(3)
13
14 print(a.balance) -- 3
15 print(b.balance) -- 3
```

```
1 -- oop-self.lua
2
3 Account = { balance = 0 }
4
5 function Account.withdraw(self, x)
6     self.balance = self.balance - x
7 end
8
9 a = {balance = 0; withdraw = Account.withdraw}
10 b = {balance = 0; withdraw = Account.withdraw}
11
12 a.withdraw(a, 20)
13 b.withdraw(b, 10)
14
15 print(a.balance) -- -20
16 print(b.balance) -- -10
17
18 a:withdraw(6)
19 --[[ `:` er syntaktisk sukker for a.withdraw(a, 6 ) ]]]
20 print(a.balance) -- -26
21 print(b.balance) -- -10
```

```
1 -- oop-constructor.lua
2
3 Account = { balance = 0 }
4
5 function Account.withdraw(self, x)
6     self.balance = self.balance - x
7 end
8
9 --[[ Constructor ]]]--
10 function Account:new(o)
11     o = o or {}
12     -- Det nye objekt arver sine metoder fra Account
13     -- via __index metoden
14     setmetatable(o, self)
15     self.__index = self
16     return o
17 end
18
19 c = Account:new{ balance = 75 }
20 c:withdraw(50)
21 print(c.balance) -- 25
```

# Weak tables

- Lua har automatisk memory management og garbage collection (GC)
- Selv den smarteste GC kan ikke altid vide hvad der er garbage, derfor weak tables
- Det er kun objekter der GC
- collectgarbage() kører en GC

# Weak tables

Tabels kan have forskellige modes

Sættes ved at ændre værdien for feltet `__mode` for tabellens metatable

- Strong – default, betyder der ikke GC hvis tabellen er accessible
- Weak key – weak reference på keys
- Weak value – weak reference på values
- Weak – weak reference af både key og values

```
mt.__mode = "k"      - "k", "v" eller "kv"
```

```
1 -- weak-table-key.lua
2
3 a, b = {}, {}
4
5 setmetatable(a, b)
6 b.__mode = "k"
7
8 key = {} -- key table: 0xda1750
9 a[key] = 1
10
11 key = {} -- key table: 0xda19a0
12 a[key] = 2
13
14 collectgarbage()
15
16 for k, v in pairs(a) do
17     print(v)
18 end
19
20 -- 2
21
22 --[[ Output hvis vi havde undladt: __mode = 'k' ]]]--
23 -- 1
24 -- 2
```

```
1 -- weak-table-value.lua
2
3 x, y = {}, {}
4
5 setmetatable(x, y)
6 y.__mode = "v"
7
8 val = {}
9 x['key'] = val
10
11 val = {}
12 x['keys'] = val
13
14 collectgarbage()
15
16 for k, v in pairs(x) do
17     print(v)
18 end
19
20 -- table: 0x1ee37a0
21
22 --[[ Hvis vi havde undladt __mode == 'v' ]]--
23 -- table: 0xa217d0
24 -- table: 0xa21640
```

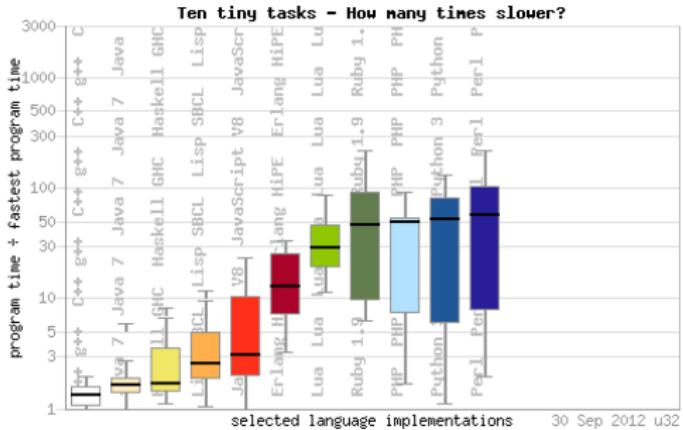
- basic – Lua basis modul: assert(), print(), tostring() etc.
- coroutine – operationer ifbm. coroutiner
- package – funktionalitet til at loade packages/biblioter
- math – forskellige matematiske funktioner
- table – funktioner til operationer på tabeller
- string – streng funktioner
- io – I/O funktioner fx læse eller skrive til/fra filer
- os – Operativsystem funktioner
- debug – til debugging og profiling
- bit32<sup>6</sup> – indeholder funktioner til bitwise manipulation

---

<sup>6</sup>Fra Lua 5.2



# Hastighed



Kilde: <http://shootout.alioth.debian.org/u32/which-programming-languages-are-fastest.php>



```
1 # primes.py
2
3 import math
4
5 def primes(n):
6     primes = [2]
7     s = 0
8     k = 0
9     for i in xrange(3, n, 2):
10         s = math.ceil(math.sqrt(i))
11         k = len(primes)
12         for j in xrange(0, k):
13             if i % primes[j] == 0:
14                 break
15             elif j+1 == k or primes[j] > s:
16                 primes.append(i)
17                 break
18     return primes
19
20 p = primes(1000000)
21 print len(p)
```

```
1 <?php
2 // primes.php
3
4 function primes($n) {
5     $primes = array();
6     $s = 0;
7     $k = 0;
8     $primes[] = 2;
9     for($i = 3; $i < $n; $i++) {
10         $s = (int)ceil(sqrt($i));
11         $k = count($primes);
12         for($j = 0; $j < $k; $j++) {
13             if(($i % $primes[$j]) == 0) {
14                 break;
15             } else if($j+1 == $k || $primes[$j] > $s) {
16                 $primes[] = $i;
17                 break;
18             }
19         }
20     }
21     return $primes;
22 }
23
24 $p = primes(1000000);
25 print(count($p) . "\n");
26 ?>
```

```
1 -- primes.lua
2
3 -- Find alle primtal op til 'n'
4 function primes(n)
5     local primes = {2}
6     local s = 0
7     local k = 0
8     for i = 3, n, 2 do
9         s = math.ceil(math.sqrt(i))
10        k = #primes
11        for j = 1, k do
12            if i % primes[j] == 0 then
13                break
14            elseif j == k or primes[j] > s then
15                primes[k + 1] = i
16                break
17            end
18        end
19    end
20    return primes
21 end
22
23 local p = primes(1000000)
24 print(#p)
```

Lenovo T410s i5@2.4GHz

```
$ time php primes.php
```

```
78498
```

```
real    0m3.616s
```

```
user    0m3.600s
```

```
sys     0m0.008s
```

```
$ time python primes.py
```

```
78498
```

```
real    0m2.533s
```

```
user    0m2.524s
```

```
sys     0m0.000s
```

```
$ time lua ./lua/primes.lua
```

```
78498
```

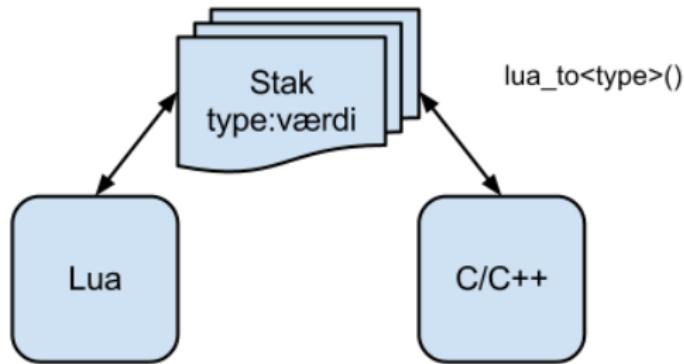
```
real    0m1.462s
```

```
user    0m1.448s
```

```
sys     0m0.008s
```

- Det er muligt at embedde en Lua fortolker i C/C++ programmer
- Det er muligt at kalde Lua funktioner fra C/C++
- Det er muligt at kalde C/C++ funktioner fra Lua
- Udfordringer:
  - Dynamic & static typed
  - Automatic & manual memory management
- Løsning: Lua & C kommunikerer via en stak

# Lua & C integration



All kommunikation går igennem stak'en: `lua_State*`

```
1 -- lua-c.lua
2
3 luainit = package.loadlib("./c/lib/mylib.a", "luaopen_mylib")
4
5 func = function(a, b)
6     return a + b
7 end
8
9 luainit()
10
11 d, h = double_and_half(10)
12 print(d, h)
13 -- 20    5
14
15 call_lua()
16 -- Lua siger svaret er 42
```

```

1 /* mylib.c */
2
3 #include <stdio.h>
4 #include "lua.h"
5
6 static int myfunc(lua_State* l) {
7     int a = lua_tointeger(l, -1);
8     lua_pushinteger(l, a * 2);
9     lua_pushinteger(l, a / 2);
10    /* Vi returnerer to resultater */
11    return 2;
12 }
13
14 static int myotherfunc(lua_State* l) {
15     int z = 0;
16     lua_getglobal(l, "func");
17     if( lua_isfunction(l, lua_gettop(l)) ) {
18         lua_pushinteger(l, 20);
19         lua_pushinteger(l, 22);
20         /* Kald funktion med 2 args og 1 resultat */
21         lua_pcall(l, 2, 1, 0);
22         z = lua_tointeger(l, -1);
23         lua_pop(l, -1);
24         printf("Lua siger svaret er %d\n", z);
25     }
26     return 0;
27 }
28
29 int luaopen_mylib(lua_State* l) {
30     lua_register(l, "double_and_half", myfunc);
31     lua_register(l, "call_lua", myotherfunc);
32     return 0;
33 }
```

```
1 # Makefile
2
3 CC=gcc
4 CFLAGS=-Wall -O2 -fPIC -pedantic -ansi -shared
5 LFLAGS=-fPIC -shared
6
7 all: ../lib/mylib.a
8
9 ../lib/mylib.a: mylib.o
10    $(CC) mylib.o $(LFLAGS) -I/usr/include/lua5.1 \
11        -L/usr/include -o ../lib/mylib.a
12
13 mylib.o: mylib.c
14    $(CC) $(CFLAGS) -I/usr/include/lua5.1 -c mylib.c -o mylib.o
15
16 clean:
17    rm -rf *.o ../lib/*.a
```

- Det er også muligt at integrere Lua med Java
  - Luajava – <http://www.keplerproject.org/luajava/>
  - JNLua – Java Native Lua  
<http://code.google.com/p/jnlua/>
- Jeg har ingen praktisk erfaring med at kombinere Java og Lua...



# LuaJit - just-in-time Compiler

- Installer via pakkesystem eller hent fra luajit.org
- State-of-the-art JIT compiler skrevet i assembler
- OS: Linux, Android, BSD, OSX, IOS, Windows
- CPU: X86, X64, ARM, PPC, MIPS
- Compilering er hurtig og giver markante hastighedsforbedringer

# LuaJit - just-in-time Compiler

Jit compilet kode

```
$time lua hello-world.lua
```

```
real    0m0.002s  
user    0m0.000s  
sys     0m0.000s
```

```
$time luajit hello-world.lua
```

```
real    0m0.002s  
user    0m0.000s  
sys     0m0.000s
```



# LuaJit - just-in-time Compiler

Jit compilet kode

```
$time lua primes.lua
78498
real    0m1.063s
user    0m1.060s
sys     0m0.000s
```

```
$time luajit primes.lua
78498
real    0m0.120s
user    0m0.116s
sys     0m0.000s
```

# LuaRocks - pakkesystem

- Hent LuaRocks<sup>7</sup> via pakkesystem  
sudo apt-get install luarocks
- Eller hent tarball <http://luarocks.org/en/Download>

```
$ ./configure  
$make  
$sudo make install
```

---

<sup>7</sup>Ruby Gems, Perl CPAN, Haskell Cabal...



# LuaRocks - pakkesystem

## Indstaller pakker

- Find den relevante pakke

<http://luarocks.org/repositories/rocks/>

```
$luarocks help
```

```
$luarocks help install
```

```
$luarocks install luasocket
```



```
package = "LuaSocket"
version = "2.0.2-5"
source = {
    url = "http://luaforge.net/frs/download.php/2664/luasocket-2.0.2.tar.gz",
    md5 = "41445b138deb7bcfe97bff957503da8e"
}
description = {
    summary = "Network support for the Lua language",
    detailed = [
        LuaSocket is a Lua extension library that is composed by two parts: a C core
        that provides support for the TCP and UDP transport layers, and a set of Lua
        modules that add support for functionality commonly needed by applications
        that deal with the Internet.
    ],
    homepage = "http://luaforge.net/projects/luasocket/",
    license = "MIT"
}
dependencies = {
    "lua >= 5.1, < 5.2"
}
```

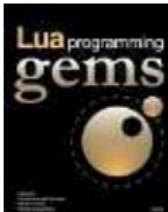
```
build = {
    type = "make",
    build_variables = {
        CFLAGS = "$(CFLAGS) -DLUASOCKET_DEBUG -I$(LUA_INCDIR)",
        LDFLAGS = "$(LIBFLAG) -O -fpic",
        LD = "$(CC)"
    },
    install_variables = {
        INSTALL_TOP_SHARE = "$(LUADIR)",
        INSTALL_TOP_LIB = "$(LIBDIR)"
    },
    platforms = {
        macosx = {
            build_variables = {
                CFLAGS = "$(CFLAGS) -DLUASOCKET_DEBUG -DUNIX_HAS_SUN_LEN \
                          -fno-common -I$(LUA_INCDIR)"
            }
        },
        windows={ ... }
    },
    copy_directories = { "doc", "samples", "etc", "test" }
}
```

# LuaRocks - pakkesystem

## Eksempler på pakker

- LuaSocket – socket library
- LuaCurl – curl library til Lua
- LuaRedis – Redis library
- hige – mustache template modul
- abelhas – Particle swarm optimization (PSO)
- lpdf – A library for generating PDF documents (PDFlib)
- LuaPSQL – PostgreSQL client bindings for Lua
- lplib – Geometriske funktioner

# Ressourcer

-  Programming in Lua
-  Lua Programming Gems
-  Lua Reference Manual



- Officiel hjemmeside <http://www.lua.org>
- Programming in Lua <http://www.lua.org/pil/>
- Side om Lua <http://lua-users.org/>
- Pakkesystem <http://luarocks.org>
- JiT compiler <http://luajit.org>
- Lua projekter <http://luaforge.net/>
- Ion windowmanager <http://tuomov.iki.fi/software/>
- Lua projekt side <http://www.keplerproject.org>
- Wikipedia [http://en.wikipedia.org/wiki/Lua\\_\(programming\\_language\)](http://en.wikipedia.org/wiki/Lua_(programming_language))
- Reimplementationer af Lua  
<http://lua-users.org/wiki/LuaImplementations>



# Links

(fortsat)

- Lua reference [http://pgl.yoyo.org/luai/i/\\_](http://pgl.yoyo.org/luai/i/_)
- Lua C reference <http://pgl.yoyo.org/luai/i/3.7+Functions+and+Types>
- World of Warcraft <http://wowprogramming.com>
- World of Warcraft <http://www.wowwiki.com/Lua>
- Lighttpd <http://redmine.lighttpd.net/projects/lighttpd/wiki/absolution>
- Redis <http://redis.io/commands/eval>
- Xavante webserver  
<http://keplerproject.github.com/xavante/>
- PostgreSQL <http://pllua.projects.postgresql.org/>
- Ginga TV Middleware <http://www.ginga.org.br/>

