Kurs języka Lua

01b – Podstawy

Jakub Kowalski

Instytut Informatyki,
Uniwersytet Wrocławski

2017

Hello World

Standalone Lua (lua5.3 hello.lua)

```
print "Hello World!" -- Welcome ;-)
```

Embedded Lua (g++ hello.cpp -I/usr/include/lua5.3 -llua5.3)

```
#include "lua.hpp"
int main(int argc, char **argv)
{
  lua_State* L = luaL_newstate();
  if (!L) return 1;
  luaL_openlibs(L);
  const char* command = "print 'Hello world!'";
  luaL_dostring(L,command);
  lua_close(L);
  return 0;
}
```

Comments

Komentarze

```
-- komentarz jednolinijkowy
-- x = 1024

--[[
komentarz
wielolinijkowy
x = 1024
--]]

-- który łatwo odkomentować
---[[
x = 1024
--]]
```

Chunks, Blocks

Chunks

- Sekwencje statementów (definicji, przypisań, pętli, ...)
- Średniki oraz zakończenia linii są opcjonalne

```
a = 1
b = a*2
a = 1; b = a*2
a = 1 b = a*2
```

Blocks

- Naturalnie – wnętrza pętli i wyrażeń warunkowych
- Sztucznie – konstrukcja `do ... end`
- Służą do ustalania zasięgu zmiennych

Scope

Zmienne globalne

- Nie wymagają specjalnej deklaracji
- Domyślnie mają wartośc `nil`
- Unikamy nazw uppercase zaczynających się od podkreślnika (Lua-reserved: _VERSION, _ENV, _G, ...)

```
print(b) --> nil
b = 10
print(b) --> 10
b = nil -- 'kasujemy' zmienną globalną
print(b) --> nil
```

Scope

Zmienne lokalne

- Szybkiego dostępu
- Nie zaśmiecają przestrzeni nazw

```
a = 10          -- zmienna globalna
local b = 11 -- zmienna lokalna wobec bloku

x = 10          -- globalna
if ... then
  local x     -- lokalna wobec then..else
  x = 20      -- nadpisujemy zmienną lokalną
  print(x) --> 20 (lokalny x)
else
  print(x) --> 10 (globalny x)
end
print(x) --> 10 (globalny x)
```

Typy

Nil

- Pojedyńcza wartość - `nil`
- Oznacza brak użytecznej wartości (niezainicjalizowane zmienne, puste pola tabeli, niepodane parametry funkcji)
- Wiele operacji na `nil` kończy się błędem

Boolean

- Przymuje wartości `true` lub `false`
- Każda inna wartość może być traktowana jako boolowska
- ale wszystko oprócz `false` i `nil` jest ewaluowane do prawdy!

Number

- Floaty podwójnej precyzji (64-bit)
- (w lua5.3 dodano subtyp Intger, który można explicite kontrolować)
- Można je zapisywać stosując notację naukową oraz szesnastkowo (używając `0x`)
- Dzielenie przez 0 to nie błąd tylko `inf`/`-inf`

String

- Napisy są niemutowalne
- Nie mają ograniczenia na długość i zawartość (mogą przechowywać dane binarne)
- Ograniczane cudzysłowami pojedyńczymi lub podwójnymi
- Escape character to \
- Funkcja `tostring` rzutuje swój argument na napis

```
print (#'ala\n?') --> 5 -- długość napisu
print ("Napis z \"cytowaniem\".")
  --> Napis z "cytowaniem".
print (tostring(10)=='10') --> true
print ('11'+1) --> 12
print ('jedenaście'+1) -->  attempt to perform
                --> arithmetic on a string value
print ('11'..1) --> 111
x = 'prefix'
y = x..'-sufix'
print (x) --> prefix
print (y) --> prefix-sufix
```

Table

- Tabele w Lua to słowniki klucz→wartość (tablice asocjacyjne)
- Kluczem może być każda wartosc oprócz `nil`
- Tablice to podstawowa (i bardzo złożona) struktura w Lua
- Zawsze operujemy na referencjach to tablic
- Tablice mają specjalne możliwości jeśli indeksujemy je od 1!

```
a = {} -- tworzymy pustą tablicę (konstruktor)
-- tworzymy częściowo wypełnioną tablicę (od 1)
b = {10, 11, 'dwanaście'}
print (b[1]) --> 10
-- długość fragmentu tablicy indeksowanago od 1!
print (#a) --> 0
b['ala'] = 'kot'
print (b['ala']) --> kot
print (#b) --> 3 (!!!)
b[4] = 'trzynaście'
print (#b) --> 4
c = b
print (c[3]) --> dwanaście
```

Function

- Funkcje w Lua są wartościami pierwszego rzędu
- Można je przechowywać w zmiennych, tabelach, przekazywać jako parametry funkcji i zwracać w funkcjach
- Więcej o funkcjach później

Userdata

- Ustalony blok pamięci
- Pozwala na przechowywanie w zmiennych Lua dowolnych danych z C

Thread

- Właściwie *coroutines*
- Specjalny typ do reprezentacji wątków

Operatory

Operatory arytmetyczne

- `a + b` (dodawanie)
- `a - b` (odejmowanie)
- `a * b` (mnożenie)
- `a / b` (dzielenie)
- `a % b` (modulo, i.e. `a-floor(a/b)*b`)
- `a ^ b` (potęgowanie)
- `- a` (negacja)

Operatory relacyjne

- `== ~=`
- Porównania działają dla dowolnych typów.
- Wartości dwóch różnych typów są zawsze różne
- Tabele, userdata i funkcje porównywane po referencji
- `< > <= >=`
- Porównania tylko na parze liczb lub napisów (wtedy leksykograficznie)

Operatory logiczne

- `a and b` (koniunkcja)
- `a or b` (alternatywa)
- Leniwa ewaluacja
- Zwraca ostatnio zewaluowany argument!
- `not a` (negacja)
- Zwraca zawsze prawdę lub fałsz

```
print(4 and 5) --> 5
print(nil and 13) --> nil
print(nil or 13) --> 13
```

Konkatenacja

- Konkatenacja napisów: `..`
- Automatycznie konwertuje liczby na napisy

```
print('Hello ' .. "World") --> Hello World
print('Number ' .. 2 .. 7) --> Number 27
```

Instrukcje warunkowe

```
if a < 0 then a = -a end

if a < 0 then
  print ('negative a')
else
  print ('positive a')
end

if op == "+" then
  r = a + b
elseif op == "-" then
  r = a - b
elseif op == "*" then
  r = a * b
else
  error("invalid operation")
end
```

Pętle while, repeat

```
local i = 1
while a[i] do
  print(a[i])
  i = i + 1
end

line = ""
repeat
  line = os.read()
until line ~= ""
print(line)
-- lokalne zmienne zadeklarowane w bloku
-- repeat-until są widoczne w warunku pętli!
```

Pętla for (numeryczna)

```
sum = 0
for i=1, 5+sum do -- ewaluowane raz, na początku
  sum = sum + (2* i-1) -- 'i' lokalna, read-only
end
print(sum)
print (i) --> nil

sum = 0
for i=5,1,-1 do -- start, end (inclusive!), step
  sum = sum + (2* i-1)
end
print(sum)

-- pętla for ma również wersję generyczną,
-- o której opowiemy sobie za tydzień
```

Funkcje

```
local function increment (n)
  return n+1
end

function table_sum(tab)
  local sum = 0
  for i=1,#tab do
    sum = sum + tab[i]
  end
  return sum
end

local f = increment
print (f(666)) --> 667
```

break i return

```
-- break i return muszą być ostatnimi
-- instrukcjami w swoim bloku
local i = 1
while a[i] do
  if a[i] == v then break end
  i = i + 1
end

function foo ()
  return -- błąd!
  do return end -- obejście - OK!
  ... -- kod który się nie wykona
end

-- nie ma continue!
-- (ale jest goto)
```

Dziękuję za uwagę

Za tydzień:
głębiej o tablicach i funkcjach,
uzupełnienie pominiętych szczegółów,
trochę przydatnych tricków.