Kurs języka Lua

04 – Czas na wzorce

Jakub Kowalski

Instytut Informatyki,
Uniwersytet Wrocławski

2017

Plan na dzisiaj

Opowiemy sobie dokładniej o

1. Dacie i czasie
2. Dopasowywaniu wzorców

Data i czas

Reprezentacja

Pojedyncza liczba

Liczba sekund od ustalonej daty (*epoch*)
POSIX i Windows: `Jan 01, 1970, 0:00 UTC`

Tablica

```
{ year = 1998,
  month = 9,
  day = 16,
  yday = 259, -- dzień roku (1 to 1 stycznia)
  wday = 4,   -- dzień tygodnia (środa)
  hour = 23,
  min = 48,
  sec = 10,
  isdst = false, -- daylight saving?
} -- nie ma informacji o strefie czasowej
```

Czas

os.time ([table])

```
-- bez argumentów zwraca aktualny czas (integer)
os.time() --> 1489481447
os.time() --> 1489481447

-- Dla argumentu będącego tablicą konwertuje
-- datę na format liczbowy
print (os.time{year=2015, month=8, day=15,
               hour=12, min=45, sec=20})
--> 1439635520
```

Data

os.date ([format [, time]])

W pewnym sensie odwrotność os.time.
Zamienia liczbową reprezentację na bardziej skomplikowane formaty.

```
-- formatowanie '*t' generuje tablicę:
os.date('*t', 1439635520)
--> {year=2015, day=15, month=8, hour=12, min=45,
-->  sec=20, yday=227, wday=7, isdst=true }

-- Formatowania korzystają z dyrektyw postaci
-- %<litera>
os.date('%d/%m/%Y', 1439635520) --> 15/08/2015
-- domyślnie argument  time to aktualny czas
os.date('a %A in %B') --> a Tuesday in March
```

Formatowanie daty

```
print (os.date('%Y-%m-%dT%H:%M:%S', 906000490))
--> 1998-09-17T04:48:10
print (os.date('%c', 0)) --> 01/01/70 01:00:00

local t = os.date('*t') -- get current time
os.date('%Y/%m/%d', os.time(t)) --> 2017/03/14
t.day = t.day + 40
os.date('%Y/%m/%d', os.time(t)) --> 2017/04/23

-- konwersja czasu do tablicy normalizuje go
print (t.day, t.month)          -->  14   3
t.day = t.day - 40
print (t.day, t.month)          --> -26   3
t = os.date('*t', os.time(t))
print (t.day, t.month)          -->   2   2
```

Operacje na datach

os.difftime (t2, t1)

```
-- liczba dni między wydaniem Lua5.3 a Lua 5.2
local t53 = os.time{year=2015, month=1,  day=12}
local t52 = os.time{year=2011, month=12, day=16}
local d = os.difftime(t53, t52)
print (d, d//(24*3600)) --> 97027200.0   1123.0

T = {year=2000, month=1, day=1, hour=0}
myepoch = os.time(T)
now = os.time{year=2017, month=3, day=21}
diff = os.difftime(now, myepoch)
T.sec = diff
print (os.date('%Y/%m/%d', os.time(T)))
--> 2017/03/21
```

Mierzenie czasu

os.clock ()

- Używany w celu mierzenia np. czasu działania programu
- Zwraca przybliżenie czasu procesora w sekundach
- Dokładność zależna od systemu

```
local x = os.clock()
print ('time till now:', x)
--> time till now:      0.001
local s = 0
for i=1, 100000000 do s = s + i end
print (string.format(
       'elapsed time: %.2f', os.clock() - x))
--> elapsed time: 1.09
```

Dopasowywanie wzorców

Funkcje

`string.find (s, pattern [, init [, plain]])`

Zwraca indeks początkowy i końcowy pierwszego dopasowania wzorca w tekście.

`string.match (s, pattern [, init])`

Zwraca fragment napisu który jako pierwszy dopasował się do wzorca.

`string.gsub (s, pattern, repl [, n])`

Dokonuje w napisie podstawień za zadany wzorzec.

`string.gmatch (s, pattern)`

Zwraca funkcję która iteruje po wszystkich wystąpieniach wzorca w napisie.

Wzorce

Klasy znaków

- `.` – dowolny znak
- `%a` – litera – `[A-Za-z]`
- `%c` – znak kontrolny – `[%z\001-\031\127]`
- `%d` – cyfra – `[0-9]`
- `%g` – drukowalny znak (poza spacją)
- `%l` – mała litera – `[a-z]`
- `%p` – znak interpunkcyjny – `[!-/:-@%[\\%]^_‘{}~|]`
- `%s` – biały znak – `[\t-\r]`
- `%u` – duża litera – `[A-Z]`
- `%w` – znak alfanumeryczny – `[A-Za-z0-9]`
- `%x` – cyfra szesnastkowa – `[0-9A-Fa-f]`
- `%z` – pozwala reprezentować znak z kodem `\000`
- `%<znak>` – reprezentuje ten znak (magic characters escape)
- `[<set>]` – reprezentuje unię znaków w zbiorze
- `[^<set>]` – reprezentuje dopełnienie zbioru znaków
- `%<UPPERCASE_MAGIC>` – dopełnienie klasy "małej litery"

Wzorce

Pattern items

- `[<...>]` – zbiór klas (opisujących jeden znak)
- `[<...>]*` – zero lub więcej powtórzeń zbioru/klasy (greedy)
- `[<...>]+` – jedno lub więcej powtórzeń zbioru/klasy (greedy)
- `[<...>]-` – zero lub więcej powtórzeń zbioru/klasy (lazy)
- `[<...>]?` – zero lub jedno powtórzenie zbioru/klasy (greedy)
- %n – dla n w przedziale 0–9, dopasowuje się do n-tego capture
- %bxy – testuje balans między znakiem otwierającym x a zamykającym y.
- `%f[<set>]` – (frontier) dopasowuje się do pustego napisu takiego, że następny znak należy do zbioru a poprzedni nie.

Pattern

Pattern (wzorzec) jest sekwencją pattern items.

- `^` – na początku wzorca oznacza dopasowanie do początku napisu
- `$` – na końcu wzorca oznacza dopasowanie do końca napisu

Przykłady

```
print(string.match('babba abb', 'ab')) --> ab

sa = 'deadline date is 04/04/2017.'
sb = '5/101/24 is a weirdo date'
date1 = '%d%d/%d%d/%d%d%d%d'
print(string.match(sa, date1)) --> 04/04/2017
print(string.match(sb, date1)) --> nil
date2 = '%d+/%d+/%d+'
print(string.match(sa, date2)) --> 04/04/2017
print(string.match(sb, date2)) --> 5/101/24

print(string.gsub('hello, up-down!', '%A', '.'))
--> hello..up.down.   4
```

Przykłady

```
-- identyfikatory Lua
'[_%a][_%w]*'

-- pusta para nawiasów
'%(%s*%)'

-- liczba całkowita z opcjonalnym znakiem
'[+-]?%d+'

-- zlicza samogłoski w tekście
_, nvow = string.gsub(text, '[AEIOUaeiou]', '')

string.find('a [word]', '[')
--> malformed pattern (missing ']')
string.find('a [word]', '[', 1, true) -- OK
```

Przykłady

```
print (string.gsub('one, and two; and three',
       '%a+', 'word'))
--> word, word word; word word

local words = {}
for w in string.gmatch(s, '%a+') do
  words[#words+1] = w
end
```

```
test = 'int x; /* x */ int y; /* y */ '
print (string.gsub(test, '/%*.*%*/', ''))
-->
```

Przykłady

```
print (string.gsub('one, and two; and three',
       '%a+', 'word'))
--> word, word word; word word

local words = {}
for w in string.gmatch(s, '%a+') do
  words[#words+1] = w
end
```

```
test = 'int x; /* x */ int y; /* y */ '
print (string.gsub(test, '/%*.*%*/', ''))
--> int x;    1
```

Przykłady

```
print (string.gsub('one, and two; and three',
       '%a+', 'word'))
--> word, word word; word word

local words = {}
for w in string.gmatch(s, '%a+') do
  words[#words+1] = w
end
```

```
test = 'int x; /* x */ int y; /* y */ '
print (string.gsub(test, '/%*.*%*/', ''))
--> int x;    1
print (string.gsub(test, '/%*.-%*/', ''))
--> int x;  int y;    2
```

Przykłady

```
s = '12.234.35.'
print(string.match(s, '^%d+%p')) --> 12.
print(string.match(s, '%d+%p$')) --> 35.

print(string.match(s, '^$^$')) --> s =
```

Przykłady

```
s = '12.234.35.'
print(string.match(s, '^%d+%p')) --> 12.
print(string.match(s, '%d+%p$')) --> 35.

print(string.match(s, '^$^$')) --> s = '$^'

print(string.match('<<<x < s> q<w> >', '%b<>'))
-->
```

Przykłady

```
s = '12.234.35.'
print(string.match(s, '^%d+%p')) --> 12.
print(string.match(s, '%d+%p$')) --> 35.

print(string.match(s, '^$^$')) --> s = '$^'

print(string.match('<<<x < s> q<w> >', '%b<>'))
--> <x < s> q<w> >
print(string.match('x < s> q<w> >>', '%b<>'))
-->
```

Przykłady

```
s = '12.234.35.'
print(string.match(s, '^%d+%p')) --> 12.
print(string.match(s, '%d+%p$')) --> 35.

print(string.match(s, '^$^$')) --> s = '$^'

print(string.match('<<<x < s> q<w> >', '%b<>'))
--> <x < s> q<w> >
print(string.match('x < s> q<w> >>', '%b<>'))
--> < s>
print(string.match('<<<x < s> q<w> >', 'q%b<>'))
--> q<w>

s = 'the anthem is the theme'
print(string.gsub(s, '%f[%w]the%f[%W]', 'one'))
-->
```

Przykłady

```
s = '12.234.35.'
print(string.match(s, '^%d+%p')) --> 12.
print(string.match(s, '%d+%p$')) --> 35.

print(string.match(s, '^$^$')) --> s = '$^'

print(string.match('<<<x < s> q<w> >', '%b<>'))
--> <x < s> q<w> >
print(string.match('x < s> q<w> >>', '%b<>'))
--> < s>
print(string.match('<<<x < s> q<w> >', 'q%b<>'))
--> q<w>

s = 'the anthem is the theme'
print(string.gsub(s, '%f[%w]the%f[%W]', 'one'))
--> one anthem is one theme     2
```

Captures

Grupy przechwytujące

Mechanizm pozwalający na zapamiętanie dopasowanych podwyrażeń w celu późniejszego wykorzystania.

```
pair = "name = Anna"
print ( string.match(pair, "(%a+)%s*=%s*(%a+)") )
--> name    Anna
print ( string.find(pair, "(%a+)%s*=%s*(%a+)") )
--> 1    11    name    Anna

date = "Today is 21/3/2017."
d, m, y = string.match(date, "(%d+)/(%d+)/(%d+)")
-- _, _, d, m, y = string.find(date, <...>)
print(d, m, y) --> 21 3 2017
```

Captures

Dopasowywanie kopii

Wyrażenie %*n* we wzorcu odpowiada dopasowaniu numer *n*.
Specjalne wyrażenie %0 odpowiada całemu dopasowanemu fragmentowi.

```
s = [[then he said: "it's all right"!]]
-- dlaczego "[\"']].-[\"']]" nie zadziała?
```

Captures

Dopasowywanie kopii

Wyrażenie %*n* we wzorcu odpowiada dopasowaniu numer *n*.
Specjalne wyrażenie %0 odpowiada całemu dopasowanemu fragmentowi.

```
s = [[then he said: "it's all right"!]]
-- dlaczego "[\"']).-[\"']" nie zadziała?
q, quote = string.match(s, "([\"'])(.-)%1")
--> q = " , quote = it's all right

-- Lua long strings:
'%[(=*)%[(.-)%]%1%]'

print(string.gsub("hello Lua!", "(%a)", "%1-%1"))
print(string.gsub("hello Lua!", "%a", "%0-%0"))
--> h-he-el-ll-lo-o L-Lu-ua-a!
```

Captures

```
print(string.gsub("hello Lua", "(.)(.)", "%2%1"))
-->
```

Captures

```
print(string.gsub("hello Lua", "(.)(.)", "%2%1"))
--> ehll ouLa

s = [[the \quote{task} is to \em{change} that]]
string.gsub(s, "\\(%a+){(.-)}", "<%1>%2</%1>")
the <quote>task</quote> is to <em>change</em> that

function trim (s)
  return string.gsub(s,'^%s*(.-)%s*$', '%1')
end
```

Replacements

Podstawienia

- Funkcji gsub jako trzeci argument można podać funkcję lub tablicę
- Tablica podstawia wartość dla klucza będącego pierwszym capture
- Argumentami funkcji są captures, a podstawiany jest jej rezultat
- Jeśli funkcja lub tablica zwrócą nil, capture zostaje bez zmian

```
function expand (s)
  return (string.gsub(s, "$(%w+)", _G))
end
name = "Lua"; status = "great";
print(expand("$name is $status, isn't it?"))
--> Lua is great, isn't it?
local othername = 'C++'
print(expand("$othername is $status, isn't it?"))
--> $othername is great, isn't it?
```

Replacements

```
function expand (s)
  return (string.gsub(s, "$(%w+)",
     function (n) return tostring(_G[n]) end))
end
print(expand("print = $print; a = $a"))
--> print = function: 68d16910; a = nil

s = 'ala ma 2 koty'
words = {}
string.gsub(s, "(%a+)",
   function (w) table.insert(words, w) end)
--> words = {'ala','ma','koty'}
```

Przykłady

Zagnieżdżone komendy LaTeX→ XML

```
function toxml (s)
  s = string.gsub(s, '\\(%a+)(%b{})',
    function (tag, body)
      body = string.sub(body,2,-2)
      body = toxml(body)
      return ('<%s>%s</%s>'):format(tag,body,tag)
    end)
  return s
end

print (toxml('\\title{The \\bold{big} example}'))
--> <title>The <bold>big</bold> example</title>
```

Przykłady

Pusty capture

'Pusty' capture () posiada specjalne znaczenie - zwraca swoją pozycję.

```
print(string.match('hello world', '()ll().-()l'))
--> 3   5   10
```

```
-- Chcemy pobrać fragment kończący się dolarem
--> '(.-)%$'
```

Przykłady

Pusty capture

'Pusty' capture () posiada specjalne znaczenie - zwraca swoją pozycję.

```
print(string.match('hello world', '()ll().-()l'))
--> 3   5   10
```

```
-- Chcemy pobrać fragment kończący się dolarem
--> '(.-)%$'  -- działa kwadratowo jeśli fail
--> '^(.-)%$' -- działa liniowo jeśli fail

i, j = string.find(';$% *#hello13!', '%a*')
print (i,j) -->
```

Przykłady

Pusty capture

'Pusty' capture () posiada specjalne znaczenie - zwraca swoją pozycję.

```
print(string.match('hello world', '()ll().-()l'))
--> 3   5   10
```

```
-- Chcemy pobrać fragment kończący się dolarem
--> '(.-)%$'  -- działa kwadratowo jeśli fail
--> '^(.-)%$' -- działa liniowo jeśli fail

i, j = string.find(';$% *#hello13!', '%a*')
print (i,j) --> 1   0

-- Chcemy znaleźć długie linie (>=70 znaków)
pattern = string.rep('[^\n]', 70) .. '+'
```

Przykłady

```
-- Pattern matching nie zastąpi porządnego
-- parsera! Np. '/%*.-%*/' vs
'"a /* here"; /* tricky string */'

-- generowanie patternów
function nocase (s)
    return string.gsub(s, '%a', function (c)
     return '['..c:lower()..c:upper()..']' end)
end
print (nocase('Hi there!'))
--> [hH][iI] [tT][hH][eE][rR][eE]!

-- automatyczne dodawanie escape characters
s1=s1:string:gsub('(%W)', '%%%1') -- search
s2=s2:string:gsub('(%%)', '%%%%') -- replacement
```

Dziękuję za uwagę

Za tydzień:
domknięcia,
iteratory,
...?