Дата последней сборки документа:
4 июня 2021 г. в 03:11

Заметки. Практика использования и наиболее полезные конструкции языка Lua

Подвойский А.О.

Здесь приводятся заметки по некоторым вопросам, касающимся программирования на языке Lua в контексте работы с системой компьютерной верстки LaTeX.

1. Общие замечания

Lua не нужен разделитель между идущими подряд операторами, но в принципе можно использовать точку с запятой, если хочется. Обычно точку с запятой ставят только, если требуется разделить два и более операторов, записанных в одной строке. Переводы строк не играют никакой роли в синтаксисе Lua.

```
-- определяет функция факториала
function fact(n)
  if n == 0 then
    return 1
  else
    return n * fact(n - 1)
  end
end

print("Enter a number:")
a = io.read("*n") -- считывает число
print(fact(a))
```

Для выхода из интерактивного режима и интерпретатора следует набрать управляющий символ конца файла (Ctrl+D в UNIX, Ctrl+Z в Windows) или вызвать функцию `exit` из библиотеки операционной системы – для этого нужно набрать `os.exit()`.

Выполнять куски кода в интерактивном режиме можно с помощью функции `dofile`. Например

```
-- lib.lua
function norm(x, y)
  return math.sqrt(x^2 + y^2)
end

function twice(x)
  return 2*x
end

-- интерактивная оболочка
> dofile("lib.lua")
> n = norm(3, 4) --> 5.0
```

Неопределенные переменные возвращают nil. Lua поддерживает однострочные комментраии (`--`) и блочные многострочные `--[[...--]]`

```
--[[
  print(10) -- ничего не происходит
--]]

---[[
  print(10) --> 10
--]]
```

В первом случае обычный блочный комментарий, а во втором – блок начинается с обычного однострочного комментария (--), поэтому все, код выполняется.

Тип `nil` – это тип с единственным значением, основная задача которого состоит в том, чтобы отличаться ото всех остальных значений. Lua использует nil как *нечто, не являющееся значением*, чтобы изобразить отсутствие подходящего значения.

ЗАМЕЧАНИЕ: Проверки условий считают `nil` и булево `false` ложными, а все прочие значения истинными. В частности, при проверках условий Lua считает ноль и пустую строку истинными значениями.

Тип `number` представляет *вещественные числа*, т.е. числа двойной точности с плавающей точкой (тип double в C). В Lua нет целочисленного типа (тип integer в C).

У целых чисел есть точное представление и потому нет ошибок округления.

Узнать длину строки можно с помощью *оператора длины* (#)

```
print(#"python") --> 6
```

Для определения границ строковых литералов можно использовать как одинарные, так и двойные кавычки

```
a = "a line"
b = 'another line'
c = [[
  multilines
]]
```

2. Шпаргалка

2.1. Условия и циклы

Цикл `while`

```
num = 42

while num < 50 do
  num = num + 1 -- составных операторов типа '+=' нет
end
```

Условия

```
if num > 40 then
  print("over 40")
elseif s ~= "walternate" then -- ~= это 'не равно'; проверка на равенство выполняется как в
    Python с помощью '=='
  io.write("not over 40\n") -- по умолчанию выводит в stdout
else
  -- по умолчанию переменные глобальные
  thisGlobal = 5
```

```
  -- для того чтобы сделать переменную локальной
  local line = io.read() -- читает строку из stdin
  -- строки можно склеить с помощью оператора '..'
  print("Winter is coming, " .. line)
end
```

Цикл for

```
karlSum = 0
for i = 1, 100 do -- диапазон включает и левую, и правую границу
  karlSum = karlSum + i
end

-- чтобы переменная цикла принимала значения от большего к меньшему
fredSum = 0
for j = 100, 1, -1 do fredSum = fredSum + j end
```

Цикл repeat

```
repeat
  print("The way of the future")
  num = num - 1
until num == 0
```

2.2. Функции

```
function fib(n)
  if n < 2 then
    return 1
  else
    return fib(n - 2) + fib(n - 1)
  end
end
```

Замыкания (вложенные функции) и анонимные функции

```
function adder(x)
  -- вложенная анонимная функция создается, когда вызывается adder, и запоминает x
  return function (y) return x + y end -- анонимная функция
end

a1 = adder(9)
a2 = adder(36)

print(a1(16)) --> 25
print(a2(64)) --> 100
```

Функции могут быть локальными и глобальными

```
-- одно и тоже
function f(x) return x * x end
f = function (x) return x * x end

-- и это тоже
local function g(x) return math.sin(x) end
local g; g = function (x) return math.sin(x) end
--
```

3. Таблицы

Таблицы в Lua представляют собой ассоциативные массивы. Ключи таблиц по умолчанию имеют строковый тип

```
t = {key1 = "value1", key2 = false} -- таблица; как словарь в Python

print(t.key1) --> "value1"
t.newKey = {} -- добавить новую пару "ключ-значение"
t.key2 = nil -- удалить ключ key2 из таблицы

-- можно передать значение в функцию динамически, на ходу создавая таблицу
function h(x)
  return x.key1
end

print(h{ key1 = "value" }) --> 'value'

-- итерации по таблице
for key, value in pairs(u) do
  print(key, value)
end
```

Использование таблиц как списков / массивов

```
v = {"value1", "value2", 1.21, "gigawatts"} -- здесь неявно ключами значений становятся целочисл
    енные ключи

for i = 1, #v do
  print(v[i])
end
```

4. Метатаблицы и метаметоды

Таблица может иметь метатаблицу, с помощью которой можно перегружать операторы и тем самым менять их поведение

```
f1 = {a = 1, b = 2}
f2 = {a = 2, b = 3}

metafraction = {}
function metafraction.__add(f1, f2)
  sum = {}
  sum.b = f1.b * f2.b
  sum.a = f1.a * f2.b + f2.a * f1.b
  return sum
end

setmetatable(f1, metafraction)
setmetatable(f2, metafraction)

s = f1 + f2 -- вызовет __add(f1, f2)
```

Таблицы можно расширять

```
defaultFavs = {animal = "gru", food = "donuts"}
myFavs = {food = "pizza"}
setmetatable(myFavs, {__index = defaultFavs})
eatenBy = myFavs.animal --> 'gru'
```

`__add`, `__index` и пр. называют метаметодами. Вот полный список метаметодов

- `__add(a, b)`: `a + b`,
- `__sub(a, b)`: `a - b`,
- `__mul(a, b)`: `a * b`,
- `__div(a, b)`: `a / b`,
- `__mod(a, b)`: `a % b`,
- `__pow(a, b)`: `a ^ b`,
- `__unm(a)`: `-a`,
- `__concat(a, b)`: `a .. b`,
- `__len(a)`: `#a`,
- `__eq(a, b)`: `a == b`,
- `__lt(a, b)`: `a < b`,
- `__le(a, b)`: `a <= b`,
- `__index(a, b)`: `a.b`,
- `__newindex(a, b)`: `a.b = c`,
- `__call(a, ...)`: `a(...)`.

Список литературы

1. *Иерузалимски Р.* Программирование на языке Lua, 2013. – 413 с.