Язык программирования Lua

Данные и переменные

гр. 5030102/20201
Смирнова А. П.
Грушин А. Д.

Лексические соглашения

Lua игнорирует пробелы (включая переходы на новую строку) и комментарии.

Имена (идентификаторами) в Lua могут быть любой строкой из букв, цифр и подчеркиваний, не начинающейся с цифры.

Язык Lua чувствителен к регистру символов: and — зарезервированное слово, но And и AND — два разных, допустимых имени.

Лексические соглашения

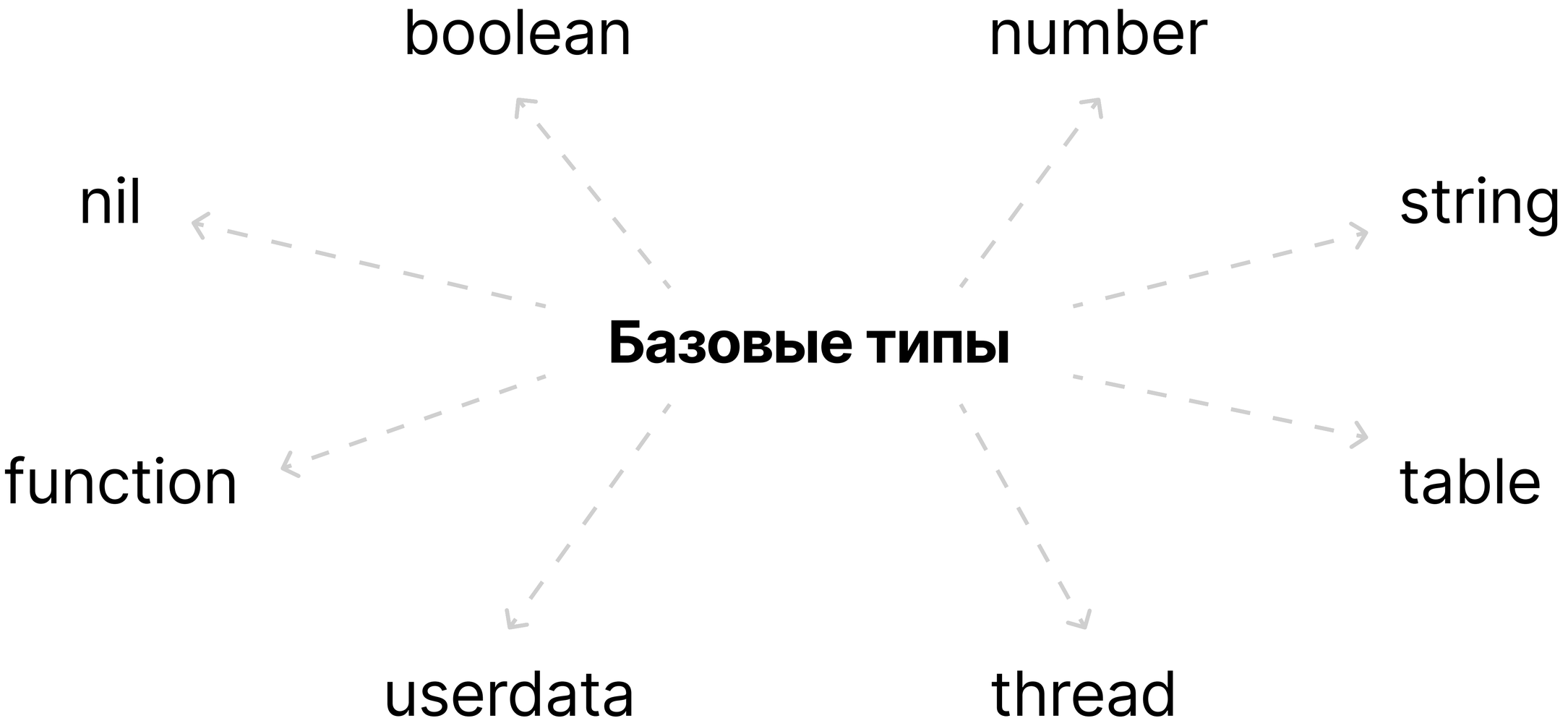
Следующие ключевые слова зарезервированы и не могут использоваться как имена:

```
and       break     do        else
elseif    end       false     for
function  goto      if        in
local     nil       not       or
repeat    return    then      true
until     while
```

Значения и типы

Lua *динамически типизированный язык*.
Это означает, что переменные не имеют типов; типы имеют только значения.

Все значения в Lua *первоклассные*.
Это означает что все значения могут быть сохранены в переменных, переданы как аргументы другим функциям, и возвращены как результаты.


boolean
number
nil
string
Базовые типы
function
table
userdata
thread

Тип *nil*

Тип *nil* (нуль) имеет одно единственное значение — **nil**.

Его главное свойство это отличаться от любых других значений, и, обычно, это означает отсутствие используемого значения.

```
local n = nil
local m   -- по умолчанию тоже nil
```

Тип *boolean*

Тип *boolean* (логический) имеет два значения — **false** (ложь) и **true** (истина).

nil и false означают ложь, а любое другое значение означает истину (даже 0 и “”)

```
local flag = true
local ok = false
```

Тип *number*

Тип *number* (число) представляет целые (integer) и вещественные (float) числа.

```
local i = 42   -- целое
local f = 3.14   -- вещественное

-- проверка подтипа числа:
-- print(math.type(i)) "integer"
-- print(math.type(f)) "float"
```

Тип *number*

Примеры допустимых целых чисел:

```
3    345    0xff    0xBEBADA
```

Примеры допустимых вещественных чисел:

```
3.0             3.1416          314.16e-2
0.31416E1       34e1 0x0.1E     0xA23p-4
0X1.921FB54442D18P+1
```

Тип *string*

Тип *string* (строка) представляет неизменные последовательности байт.

Строки в Lua могут содержать любое 8-битное значение, включая нули ('\0'). Также Lua не навязывает кодировку: чаще используют UTF-8, но для Lua это просто байты.

```
local s1 = "привет"
local s2 = 'lua'
local s3 = [[многострочная строка]]
```

Тип *string*

Литеральные строки могут быть ограничены ‘...‘ или “...“, длинные — [[...]] и [[=[[...]]=]], и могут содержать C-подобные управляющие последовательности:

```
'\a'    '\b'    '\f '   '\n'    '\r'
'\t'    '\v'    '\\'    '\"'    '\''
```

Управляющая последовательность `'\z'` пропускает следующий за ней диапазон пробелов, включая переходы строки.

Любой байт в литеральной строке возможно описать его числовым значением. Это может быть сделано с помощью управляющей последовательности `\xXX`, где XX - это пара шестнадцатиричных цифр, или с помощью `\ddd`, где ddd - последовательность до трех десятичных цифр.

Тип *string*

Когда открывается длинная скобка с непосредственным переводом строки за ней, перевод строки не включается в результирующую строку.

Например, в системе использующей ASCII, пять литеральных строк ниже кодируют одинаковые строки:

```
a = 'alo\n123"'
a = "alo\n123\""
a = '\97lo\10\4923"'
a = [[alo
123"]]
a = [==[
alo
123"]==]
```

Тип *function*

Lua может вызывать и манипулировать функциями написанными на Lua и на C.

Оба типа функций в Lua представлены типом *function* (функция).

```
local function add(a, b)
    return a + b
end
```

```
local mul = function(a, b)
    return a * b
end
```

```
local function add(a, b) return a + b end
```

Тип *userdata*

Тип *userdata* (пользовательские данные) предназначен для хранения произвольных C данных в Lua переменных.

Значение userdata представляет блок памяти (raw memory). Существуют два типа пользовательских данных: *full userdata* (полные пользовательские данные) - объект с блоком памяти, которым управляет Lua, и *light userdata* (лёгкие пользовательские данные) - простой C указатель.

Тип *thread*

Тип *thread* (поток) представляет независимый поток выполнения и используется для реализации сопрограмм (coroutine).

Lua поддерживает сопрограммы на всех системах, даже на тех, где это не поддерживается операционной системой.

Тип *table*

Тип *table* (таблица) реализует ассоциативные массивы, это значит, что массив может быть проиндексирован не только числами, но и любым Lua значением, кроме nil и NaN.

Таблицы могут быть *гетерогенными* (разнородными); т.е. могут содержать значения всех типов (кроме nil).

Присваивание nil полю таблицы удаляет соответствующий ключ.

Тип *table*

```
local t1 = {}   -- пустая
local t2 = {1, 2, 3}   -- как массив
local t3 = { x = 10, y = 20 }
   -- как запись
local t4 = { [true] = "ok", [3.14] = "pi" }
   -- произвольные ключи
```

Таблицы, **функции**, **потоки** и **пользовательские данные** (userdata) - это *объекты*: переменные фактически *не содержат* их значений, только *ссылки* на них.

Присвоение, передача параметров и возврат из функций всегда манипулируют ссылками на эти значения; эти операции не подразумевают никакого типа копирования.

Операторы

Lua поддерживает следующие **арифметические** операторы:

```
+: сложение                    -: вычитание
*: умножение                   /: деление
//: целочисленное деление     %: модуль
^: возведение в степень        -: одноместный минус
```

Lua поддерживает следующие **битовые** операторы:

```
&: битовое И                   |: битовое ИЛИ
~: битовое ИЛИ НЕ              >>: правый сдвиг
<<: левый сдвиг
~: одноместное битовое НЕ
```

Операторы

Lua поддерживает следующие операторы **сравнения**:

```
==: равенство                    ~=: неравенство
<: меньше                        >: больше
< =: меньше или равно
> =: больше или равно
```

Lua поддерживает следующие **логические** операторы:

```
10 or 20                ⟶ 10
10 or error()           ⟶ 10
nil or "a"              ⟶ "a"
nil and 10              ⟶ nil
false and error()       ⟶ false
false and nil           ⟶ false
false or nil            ⟶ nil
10 and 20               ⟶ 20
```

Приоритет

Приоритет операторов в Lua представлен далее, с меньшего к большему приоритету:

```
or
and
<     >     ≤     ≥     ~=    ==
|
~
&
<<    >>
..
+     -
*     /     //    %
unary operators (not   #     -     ~)
^
```

Приведения и преобразования

Битовые операторы всегда конвертируют вещественные операнды в целые.

Возведение в степень и деление всегда конвертируют целые операнды в вещественные.

Все остальные арифметические операции, примененные к смешанным числам (целые и вещественные), преобразуют целые числа в вещественные.

Lua также конвертирует строки в числа, когда ожидается число.

Явные функции: `tonumber()`, `tostring()`

Переменные

Переменные - это место где хранятся значения. Существует три вида переменных: глобальные переменные, локальные переменные и поля таблиц.

```
var ::= Name
```

Любое имя переменной предполагается глобальным, пока явно не определено локальным.

```
var ::= local Name
```

Перед первым присваиванием переменной, её значение равно nil.

Правила видимости

Lua - язык с лексическими областями видимости.

Область видимости локальной переменной начинается с первого выражения после её определения и продолжается до последнего не пустого выражения внутреннего блока, включающего определение.

Правила видимости. Пример

```
x = 10                          -- глобальная
do                              -- новый блок
  local x = x                   -- новый 'x'
  print(x)                      -- 10
  x = x+1
    do                            -- другой блок
      local x = x+1               -- другой 'x'
      print(x)                    -- 12
    end
  print(x)                      -- 11
end
print(x)                        -- 10
```

Владение и передача владения

Для типов nil, boolean, number, string присваивание/передача = копия значения.

Для типов table, function, thread, full userdata присваивание/аргументы/возврат = передача ссылки, без копирования.

Памятью объектов управляет GC (сборщик мусора); объект собирается, когда на него нет ссылок.

Источники

При создании этой презентации использовалась информация из документации с официального сайта языка Lua (https://www.lua.org).