LabLua
programming language research

Programming in Lua – Types

Fabio Mascarenhas

http://www.dcc.ufrj.br/~fabiom/lua

Lua Types

- Lua values come in one of eight basic types (or *tags*, to be pedantic):
  - **nil** (just `nil`), **boolean** (`true` and `false`), **number** (double precision floating point), **string** (immutable byte vectors, including \0, in whatever encoding you like)**, table** (associative arrays), **function** (named and anonymous), **userdata** (opaque blobs handled by external libraries), and **thread** (actually coroutines)
- The built-in function `type` gives the name of the type of any given value, as a string
- Variables do not have a fixed type, and can hold values of any type (even values of different types in the lifetime of the variable)

Nil

- The value `nil` denotes the absence of a useful value
- It is the value of:
  - Unitialized variables
  - Missing table fields
  - Missing function parameters
- Most operations on `nil` are errors

Booleans

- Relational operators produce booleans, but you can use any value in a condition or with logical operators
- Any Lua value is *true*, except for `false` and `nil`; in particular, the number `0` and the empty string “” are *true*!
- The `and` operator gives its first argument if it is *false*, otherwise it gives its second argument
- The `or` operator gives its first argument if it is *true*, otherwise it gives its second argument
- The `not` operator always returns `true` or `false`

Useful idioms

- The fact that `or` works with any value gives us an useful idiom for “optional” parameters:

```
function greeting(s)
  s = s or "Hello"
  print(s .. ", World!")
end

greeting()
greeting("你好")
```

- A combination of `and` and `or` gives us another idiom, the “ternary operator”, analogous to ?: in C:

```
function max(a, b)
  return (a > b) and a or b
end
```

Numbers

- Lua numbers are double precision floating point numbers
- You can write them like you would in C or Java, including scientfic notation and hexadecimal (with “0x”)
- There is no integer type, but double precision lets Lua represent any 32-bit integer number without rounding issues (exact representation goes up to 53 bits, in fact!)
- Lua has the usual arithmetic and relational operators: +, -, *, /, ^ (to the power of), % (modulo), <, >, <=, >=, ==, and ~= (not equal to)
- Be careful, division by zero is not an error

Strings

- A Lua string is an immutable sequence of bytes
- Any byte is ok, even zeros, so you can use strings to store any binary data
- But what about text? Then you have to pick an *encoding*, and stick to it. UTF-8 is a good choice!
- There are just a few built-in operations on strings: *concatenation* (..), *length* (#), and the relational operators (<, >, <= and >= use *lexicographic* ordering)
- The `string` library and other external libraries provide many others

String literals

- String literals can use either single or double quotes; double-quoted strings can have single quotes inside without escaping, and vice versa

- Literal strings can also have the following escape sequences:

  - \a (bell), \b (backspace), \f (form feed), \n (newline), \r (carriage return), \t (tab), \v (vertical tab), \\ (backslash), \" (double quote), \' (single quote), *ddd* (byte with decimal value *ddd*), \x*hh* (byte with hexadecimal value *hh*), \z (skips all spaces following it)

  - The latter three escape sequences are useful for encoding binary data!

Long strings

- You can also write *long string literals*, using `[[` and `]]` to delimit the literal

- Long strings do not interpret escape sequences, and can go for several lines; if the first character is a newline it is discarded

```
escapes = [[
  \f (form feed)
  \n (newline)
  \r (carriage return)
]]
print("|" .. escapes .. "|")
```

- You can write long strings (and long comments) with other delimiters: `[=[` and `]=]`, `[==[` and `]==]`, `[===[` and `]===]`, etc.

Tables

- Lua tables are *associative arrays*; they associate a *key* with a *value*
- Any Lua value, except `nil`, can be a key, but keys are usually numbers or strings
- Tables are a very flexible and sophisticated data structure in Lua, and can represent arrays, structs, abstract data types, objects (in the OO sense), modules…
- We will see much more of them later!

Functions

- Functions are values in Lua; this means that Lua code can store functions in variables, pass them as arguments, and return them as results
  - In particular, Lua gets a lot of use from functions as table values: this is the bedrock of both object oriented programming and Lua modules
- They also have a peculiarity not usually present in other languages: Lua functions can return multiple values
- We will also see much more of functions later

Quiz

- How can you check whether a value is a boolean without using the `type` function?