Lua

Cheatsheet

P J

Contents

1 Lua Cheatsheet

Lua is a powerful, fast, lightweight and embeddable programming language.

It is used by many frameworks, games and other applications. While it can be used by itself, it has been designed to be easy to embed in another application. It is implemented in ANSI C, a subset of the C programming language that is very portable, which means it can run on many systems and many devices where most other scripting languages would not be able to run.

1.1 Hello, World!

We can use `print()` function:

```
print("Hello, World!")
```

or `io.write()` method:

```
io.write("Hello, World!\n")
```

1.2 Comments

A comment is a code annotation that is ignored by the programming language.

There are some way of creating comments in Lua:

```
-- this is a comment
print ("Hello") -- another comment
```

These comments are called short (single-line) comments. It is also possible to create long comments, which start with a long bracket and can continue on many lines:

```
--[[
    This is a multi-line comment
]]
io.write("Just a simple test\n")
--[==[
This is a comment that contains
```

```
a closing long bracket of level 0 which is here:
]]
However, the closing double bracket doesn't make the comment end,
because the comment was opened with an opening long bracket of level 2,
and only a closing long bracket of level 2 can close it.
]==]
```

1.3 Syntax

Lua is a dynamically typed language intended for use as an extension language or scripting language.

Statements and expressions can be respectively compared to sentences and words. Expressions are pieces of code that have a value and that can be evaluated, while statements are pieces of code that can be executed and that contain an instruction and one or many expressions to use that instruction with.

For example, `3 + 5` is an expression and `variable = 3 + 5` is a statement that sets the value of variable to that expression.

```lua
-- Different types
local x = 10 -- number
local name = "Neo" -- string
local is_alive = false -- boolean
local a = nil -- no value or invalid value
```

1.4 Obtaining Lua

Lua can be obtained on the official Lua website, on the download page.

2 Expressions

Expressions are pieces of code that have a value and that can be evaluated. They cannot be executed directly (with the exception of function calls), and thus, a script that would contain only the following code, which consists of an expression, would be erroneous:

```
3 + 5
-- The code above is erroneous because all it contains is an expression.
-- The computer cannot execute '3 + 5', since that does not make sense.
```

Code must be comprised of a sequence of statements. These statements can contain expressions which will be values the statement has to manipulate or use to execute the instruction.

Some code examples in this chapter do not constitute valid code, because they consist of only expressions. In the next chapter, statements will be covered and it will be possible to start writing valid code.

2.1 Types

Lua is a dynamically typed language, so the variables don't have types, only the values have types.

But we have types for the values. There is a function called 'type' that enables us to know the type of the variable.

```
print(type("What is my type"))   --> string
t = 10

print(type(5.8 * t))             --> number
print(type(true))                --> boolean
print(type(print))               --> function
print(type(nil))                 --> nil
print(type(type(ABC)))           --> string
```

By default, all the variables will point to nil until they are assigned a value or initialized. In Lua, zero and empty strings are considered to be true in case of condition checks.

The list of data types for values are given below.

Value Type	Description
`nil`	Used to differentiate the value from having some data or no(nil) data.
`boolean`	Includes true and false as values.
`number`	Represents real(double precision floating point)
`string`	Represents array of characters.
`function`	Represents a method that is written in C or Lua.
`userdata`	Represents arbitrary C data.
`thread`	Represents independent threads of execution and
`table`	Represent ordinary arrays, symbol tables, sets, records, graphs, trees, etc., and implements associative arrays. It can hold any value (except nil).

2.1.1 Numbers

Numbers generally represent quantities, but they can be used for many other things. The number type in Lua works mostly in the same way as real numbers.

Numbers can be constructed as integers, decimal numbers, decimal exponents or even in hexadecimal. Here are some valid numbers:

- `3`
- `3.0`
- `3.1416`
- `314.16e-2`
- `0.31416E1`
- `0xff`
- `0x56`

```lua
-- The Lua
local a = 1
local b = 2
local c = a + b
```

```lua
print(c) -- 3

local d = b - a
print(d)

local x = 1 * 3 * 4 -- 12
print(x)

local y = (1+3) * 2 -- 8
print(y)

print(10 / 2) -- 5
print(2 ^ 2) -- 4
print(5 % 2) -- 1
print(-b) -- -2

-- Incerment
local level = 1
level = level + 1
print(level)
```

Arithmetic operations

The operators for numbers in Lua are the following:

Operation	Syntax	Description	Example
Arithmetic negation	`-a`	Changes the sign of a and returns the value	`-3.14159`
Addition	`a + b`	Returns the sum of `a` and `b`	`5.2 + 3.6`
Subtraction	`a - b`	Subtracts b from a and returns the result	`5.2 + 3.6`
Multiplication	`a * b`	Returns the product of a and b	`3.2 * 1.5`
Exponentiation	`a ^ b`	Returns a to the power b, or the exponentiation of a by b	`5 ^ 2`
Division	`a / b`	Divides a by b and returns the result	`6.4 / 2`
Modulus operation	`a % b`	Returns the remainder of the division of a by b	`5 % 3`

Integers

A new subtype of numbers, integers, was added in Lua 5.3. Numbers can be either integers or floats. Floats are similar to numbers as described above, while integers are numbers with no decimal part.

Float division (`/`) and exponentiation always convert their operands to floats, while all other operators give integers if their two operands were integers. In other cases, with the exception of the floor division operator (`//`) the result is a float.

2.1.2 Nil

Nil is the type of the value nil, whose main property is to be different from any other value; it usually represents the absence of a useful value. A function that would return nil, for example, is a function that has nothing useful to return.

2.1.3 Boolean

A boolean value can be either `true` or `false`, but nothing else. This is literally written in Lua as `true` or `false`, which are reserved keywords. The following operators are often used with boolean values, but can also be used with values of any data type:

Operation	Syntax	Description
Boolean negation	`not a`	If a is false or nil, returns true. Otherwise, returns false.
Logical conjunction	`a and b`	Returns the first argument if it is false or nil. Otherwise, returns the second argument.
Logical disjunction	`a or b`	Returns the first argument if it is neither false nor nil. Otherwise, returns the second argument.

Essentially, the `not` operator just negates the boolean value (makes it false if it is true and makes it true if it is false), the `and` operator returns true if both are true and false if not and the `or` operator returns true if either of arguments is true and false otherwise.

```lua
local is_alive = true
print(is_alive) -- true

local is_alive = false
print(is_alive) -- false
```

2.1.4 Strings

Strings are sequences of characters that can be used to represent text. They can be written in Lua by being contained in double quotes, single quotes or long brackets (it should be noted that comments and strings have nothing in common other than the fact they can both be delimited by long brackets, preceded by two hyphens in the case of comments).

Strings that aren't contained in long brackets will only continue for one line. Because of this, the only way to make a string that contains many lines without using long brackets is to use escape sequences. This is also the only way to insert single or double quotes in certain cases.

1. `' '`
2. `" "`
3. `[[ ]]`

```lua
local phrase = [[My name is ]]
local name = 'P J'
print(phrase .. name) -- My name is P J

-- Strings and Numbers
local age = 21
local name = "Billy"
print(name .. " is " .. age .. " Years old")
```

Escape sequence characters are used in string to change the normal interpretation of characters.

For example, to print double inverted commas (`""`), we have to use `\"` in the string.

The escape sequence and its use is listed below in the table.

Escape Sequence	Use
`\a`	Bell
`\b`	Backspace

Escape Sequence	Use
\f	Formfeed
\n	New line
\r	Carriage return
\t	Tab
\v	Vertical tab
\\	Backslash
\"	Double quotes
\'	Single quotes
\[	Left square bracket
\]	Right square bracket

It is possible to get the length of a string, as a number, by using the unary length operator (#):

```
print(#("This is a string")) --> 16
```

Concatenation

In formal language theory and computer programming, **string concatenation** is the operation of joining two character strings end-to-end. For example, the concatenation of "snow" and "ball" is "snowball".

The string concatenation operator in Lua is denoted by two dots (..).

```
print("snow" .. "ball") --> snowball
```

2.1.5 Other types

The four basic types in Lua (numbers, booleans, nil and strings) have been described in the previous sections, but four types are missing: functions, tables, userdata and threads.

- *Functions* are pieces of code that can be called, receive values and return values back.
- *Tables* are data structures that can be used for data manipulation.
- *Userdata* are used internally by applications Lua is embedded in to allow Lua to communicate with that program through objects controlled by the application.
- *Threads* are used by coroutines, which allow many functions to run at the same time.

2.2 Literals

Literals are notations for representing fixed values in source code. All values can be repre sented as literals in Lua except **threads** and **userdata**.

String literals (literals that evaluate to strings), for example, consist of the text that the string must represent enclosed into single quotes, double quotes or long brackets.

Number literals, on the other hand, consist the number they represent expressed using decimal notation (ex: `12.43`), scientific notation (ex: `3.1416e-2` and `0.31416E1`) or hexadecimal notation (ex: `0xff`).

2.3 Coercion

Coercion is the conversion of a value of one data type to a value of another data type. Lua provides automatic coercion between string and number values. Any arithmetic operation applied to a string will attempt to convert this string to a number.

Conversely, whenever a string is expected and a number is used instead, the number will be converted to a string.

This applies both to Lua operators and to default functions (functions that are provided with the language).

```
1 print("122" + 1) --> 123
2 print("The number is " .. 5 .. ".") --> The number is 5.
```

Coercion of numbers to strings and strings to numbers can also be done manually with the `tostring` and `tonumber` functions.

The former accepts a number as an argument and converts it to a string, while the second accepts a string as an argument and converts it to a number (a different base than the default decimal one can optionally be given in the second argument).