Cheatography

Lua Scripting 5.1 Cheat Sheet

by SrGMC via cheatography.com/62984/cs/16090/

Types

number
string
boolean
table
function
userdata
thread
nil

Variable type can be obtained with **type(variable)**

Note: Table index starts at 0, but can be extended to 0 or negative numbers

Arithmetic Expressions

Sum	+
Negation/Subtraction	-
Product	*
Division	/
Modulo	%
Power	^

Relational Expressions

Equal to	==
Not equal to	~=
Less than	<
Greater than	>
Less than or equal to	<=
Greater than or equal to	>=

Logical Operators

not
and
or

Even though Lua does not have a Ternary operator (condition ? truevalue : falsevalue), we can use *and* and *or* to achieve a similar effect:

```
value = (condition and truevalue) or falsevalue
```

In this case *and* returns truevalue when the condition is true and falsevalue otherwise

Tables

Tables are used with the table[key] syntax

Example:

> t = {foo="bar"} -- Same as t={["foo"]="-bar"}

> t.foo

bar

They can also be used as arrays

a = {1, 2, 3}

But in this case, index starts at 1

a = {[0]=1, [1]=2}

Tables can be extended to index 0 or even negative numbers

Table size can be found with:

> a = {1, 2, 3}

> # a

3

Functions and modules

Functions

value = **function**(*args*) body **end**

function functionName(*args*) body **end**

Functions can be used as arguments:

function f(*f2, arg1*) f2(arg1) **end**

Return skips other code below it

Modules

A common module declaration usually is:

local mymodule = {}

function mymodule.foo() **print**(*"bar"*) **end**

return mymodule

As tables can have functions assigned to a key.

To import it, just do:

> module = require("mymodule")

> module.foo()

bar

Also, you can make private functions by putting local in front of the function declaration.

Math Library

math.abs(number)
math.acos(radians), math.asin(radians), math.atan(radians)
math.ceil(number), math.floor(number)
math.cos(radians), math.sin(radians), math.tan(radians)
math.deg(radians), math.rad(degrees)
math.exp(number), math.log(number)
math.min(num1, num2, ...), math.max(-num1, num2, ...)
math.sqrt(number)
math.random(), math.random(upper), math.random(lower, upper)
math.randomseed(seed)
math.huge --*represents infinity*
math.pi

On trigonometric calculations, the number is expressed as radians.

On math.random() lower and upper are inclusive.

math.huge can be also represented with -math.huge

Control Structures

if/else statement

if (condition1) then

block

elseif (condition2) then

block

else

block

end

while loop

while (condition) do

block

end

repeat loop

Like while loop, but condition is inverted

repeat

block

until (condition)

By **SrGMC**
cheatography.com/srgmc/

Published 13th June, 2018.
Last updated 13th June, 2018.

Control Structures (cont)

Numeric for loop

```
for variable = start, stop, step do
block
end
```

Iterator for loop

```
for var1, var2, var3 in iterator do
block
end
```

Table Library

table.concat(table [, sep [, i [, j]]])	Concatenate the elements of a table to form a string. Each element must be able to be coerced into a string.
table.foreach(table, f)	Apply the function f to the elements of the table passed. On each iteration the function f is passed the key-value pair of that element in the table. Apply the function f to the elements of the table passed. On each iteration the function f is passed the key-value pair of that element in the table. *Deprecated*

Table Library (cont)

table.foreachi(table, f)	Apply the function f to the elements of the table passed. On each iteration the function f is passed the index-value pair of that element in the table. This is similar to table.foreach() except that index-value pairs are passed, not key-value pairs. *Deprecated*
table.sort(table [, comp])	Sort the elements of a table in-place. A comparison function can be provided to customise the element sorting. The comparison function must return a boolean value specifying whether the first argument should be before the second argument in the sequence.
table.insert(table, [pos,] value)	Insert a given value into a table. If a position is given insert the value before the element currently at that position.

Table Library (cont)

table.remove(table [, pos])	Remove an element from a table. If a position is specified the element at that the position is removed. The remaining elements are reindexed sequentially and the size of the table is updated to reflect the change. The element removed is returned by this function.

table.sort() example:

```
> t = { 3,2,5,1,4 }
> table.sort(t, function(a,b) return a<b end)
> = table.concat(t, ", ")
1, 2, 3, 4, 5
```

String

string.byte(s [, i [, j]])	Return the numerical code the i-th through j-th character of the string passed.
string.char(i1, i2, ...)	Generate a string representing the character numerical code passed as arguments.
string.find(s, pattern [, index [, plain]])	Find the first occurrence of the pattern in the string passed
string.format(s, e1, e2, ...)	Create a formatted string from the format and arguments provided. This is similar to the printf("format",...) function in C.

Cheatography

Lua Scripting 5.1 Cheat Sheet

by SrGMC via cheatography.com/62984/cs/16090/

String (cont)

string.gsub(s, pattern, replace [, n])	Used simply it can replace all instances of the pattern provided with the replacement. A pair of values is returned, the modified string and the number of substitutions made. The optional fourth argument n can be used to limit the number of substitutions made
string.len(s)	Return the length of the string passed.
string.lower(s)	Make all the upper case characters lower case.
string.upper(s)	Make all the lower case characters upper case.
string.match(s, pattern [, index])	Extract substrings by matching patterns.
string.rep(s, n)	Generate a string which is n copies of the string passed concatenated together.
string.reverse(s)	Reverses a string.

String (cont)

string.sub(s, i [, j])	Return a substring of the string passed. The substring starts at i. If the third argument j is not given, the substring will end at the end of the string. If the third argument is given, the substring ends at and includes j.

All functions can be used directly in string by changing string. to s:, s being the string

Example:

```
string.re ver se( " Tes t")
" Tes t":r eve rse ()"
```

Classes. Table based

```
local Person = {}
Person.__ index = Person
function Person.ne w(name,
surname)
    local self = setmet ata -
ble({}, Person)
    sel f.name = name
    sel f.s urname = surname
    return self
end
function Person.se tNa me( self,
name)
    sel f.name = name
end
function Person.ge tNa me( self)
    return self.name
end
function Person.se tSu rna me( -
self, surname)
    sel f.s urname = surname
end
function Person.ge tSu rna me( -
self)
    return self.s urname
end
return Person
-- Import with ClassName =
requir e("c las sna me")
```

Classes. Table based (cont)

> -- Use with local i = ClassName.init(-params)

Faster to create. Does not have private attributes

Classes. Closure/Instance Based

```
local function MyClass(init)
        local self = {
                pub lic _field =
0
        }
        local privat e_field =
init
        fun ction self.foo()
                return privat -
e_field
        end
        fun ction self.bar()
                pri vat e_field
= privat e_field + 1
        end
        return self
end
return MyClass
-- Import with MyClass =
requir e("M yCl ass ")
-- Use with local i = MyClas -
s(init)
```

Can have private attributes. Slower to create