Lua103 cheat sheet

Lexical conventions

- `-- foobar` : comment until end of line
- variable name: _, letters, numbers (cannot start with a number)
- reserved names: `and break do else elseif end false for function if in local nil not or repeat return then true until while`

Types

Type	Literal
number	`123 -2 45.024 9.9e6`
string	`"Foo" 'Bar'`
boolean	`true false`
nil	`nil`
table	`{}`

Operators

Number operator	Meaning
`a + b`	addition
`a - b`	subtraction
`a * b`	multiplication
`a / b`	division
`a % b`	modulo
`a ^ b`	exponentiation
`-a`	opposite

String operator	Meaning
`a .. b`	concatenation
`#a`	length

Comparison operator	Meaning
`a == b`	equal
`a ~= b`	different
`a < b`	strictly smaller
`a > b`	strictly greater
`a <= b`	smaller or equal
`a >= b`	greater or equal

Logical operators	Meaning
`a and b`	true if both `a` and `b` are true
`a or b`	true if either `a`, `b` or both are true
`not a`	true if `a` is false

Expressions

Expression	Example
literal	`1`
variable	`foo`
operation	`1 + foo`
(expr)	`(1 + foo)`
function call	`bar(4)`
table indexing	`tab[key]`

Statements

Statement	Meaning
`var = expression`	assign value of expression to variable `var`
`var1,var2 = expr1,expr2`	assign values of expressions to variables
`if` control structure	select block of code according to condition
numeric `for` loop	repeat block of code
generic `for` loop	repeat block of code
`while` loop	repeat block of code
`repeat` loop	repeat block of code
`break`	exit current loop
function call	call function, discard return values if any
`return`	exit function
`return expr`	exit function with return value
`return expr1,expr2`	exit function with multiple return values
`local var`	declare local variable `var`
`local var = expr`	declare local variable `var` with initial value
`local var1,var2 = expr1,expr2`	declare local variables with initial values

block: sequence of statements

Control structures

```
-- execute block associated with the first expression which is true
-- if all expressions are false, execute elseblock

-- `elseif` blocks: 0 or more
-- `else` block: 0 or 1

if expression1 then
    -- block1
elseif expression2 then
    -- block2
elseif expression3 then
```

```lua
    -- block3
else
    -- elseblock
end
```

```lua
-- repeat block with variable taking values
-- start, start + step, start + 2*step, start + 3*step, etc.
-- until it goes past finish

-- `step` is optional and defaults to 1

for variable = start,finish,step do
    -- block
end
```

```lua
-- repeat block as long as condition is true

while expression do
    -- block
end
```

```lua
-- repeat block until condition is true

repeat
    -- block
until condition
```

```lua
-- iterate over index/value pairs in array

for index,value in ipairs(array) do
    -- block
end
```

```lua
-- iterate over key/value pairs in table

for key,value in pairs(tab) do
    -- block
end
```

Functions

```lua
-- definition
-- parameters: 0 or more, comma separated

function name(param1, param2, param3)
    -- block
end
```

```lua
-- call
-- arguments: 0 or more, comma separated
```

```
name(arg1, arg2, arg3)
```

Tables

array: table with consecutive integer indices only

Expression	Meaning
`{}`	empty table/array
`{"a", true, 5}`	array with initial values
`#array`	length of array
`{["Alice"] = 3, [true] = 2.4, [-4] = 3}`	table with initial values
`tab[key]`	table indexing (setting/getting)

Array functions	Meaning
`table.insert(array, value)`	insert `value` at end of `array`
`table.insert(array, i, value)`	insert `value` at position `i` in `array`
`table.remove(array)`	remove last element of `array`
`table.remove(array, i)`	remove element at position `i` in `array`
`table.concat(array, sep)`	make string of array contents with separator `sep`

String library

Function	Meaning
`string.sub(s, i, j)`	returns substring of `s` from position `i` to `j`
`string.find(s, p, i)`	returns first and last position of pattern `p` in `s`, or nil if not found, starting at position `i`
`string.match(s, p, i)`	returns the first match of pattern `p` in `s` or nil if not found, starting at position `i`
`string.gsub(s, p, repl)`	returns copy of `s` where all occurences of `p` are replaced with `repl`

In `string.find()` and `string.match()`, parameter `i` is optional and defaults to 1.

Character class in pattern	Meaning
`.`	any charater
`%a`	letters
`%d`	digits
`%l`	lower case letters
`%p`	punctuation characters
`%s`	space characters
`%u`	upper case letters
`%w`	alphanumeric characters
`^`	the beginning of the string
`$`	the end of the string
`%x`, x in `^$()%.[]*+-?`	`x` itself

Modifier in pattern	Meaning
`?`	0 or 1
`*`	0 or more (biggest match possible)
`-`	0 or more (smallest match possible)
`+`	1 or more (biggest match possible)