Lua - First Steps in Programming

Knut Lickert

March 12, 2007

This Text shows some easy Lua-command and which result they produce. Additional information or an actual version of tis document can be found at http://lua.lickert.net This document will be filled next time.

Contents

1 Literals and output in Lua

- Screen output is done with the command *print*.
- There are brackets around the parameters.
- Multiple parameter are separated by commas.
- Strings are defined with ' and ".

Test-Programm

```
print     ('1: Hello World')
print    ("2: Hello World")
```

The Result:
1: Hello World
2: Hello World

1.1 New lines

Test-Programm

```
print  (  '1: Hello\
        World'  )
print  (  "2: Hello\
        World"  )
```

The Result:
1: Hello
World
2: Hello
World

Test-Programm

```
print  (  '5: Hello\n World'  )
print  (  "6: Hello\n World"  )
```

The Result:
5: Hello
 World
6: Hello
 World

1.2 Alternative String definition

You can define Strings also with [[and]].

Test-Programm

```
1 print [[Hello World]]
```

The Result:
Hello World

Test-Programm

```
1 print [[Hello World,
2 you are so nice today.]]
```

The Result:
Hello World,
you are so nice today.

- Multiple lines are possible.

Test-Programm

```
1 print [[
2 Hello World,
3 you are so nice today.]]
```

The Result:
Hello World,
you are so nice today.

- A new-line after [[is ignored.

Test-Programm

```
1 print [[
2 Hello World,
3 how [[nice]] are you today]]
```

The Result:
Hello World,
how [[nice]] are you today

- The square braces may be nested.

1.3 Special Characters

Each character can be created with \ and the octal representation[1]of the ASCII-value.

1.3.1 Named characters

Each character can be created with \ and the following short cut.

Character	Function	Shortcut
\a	PC-Sound	BEL
\b	Backspace	BS
\f	Form Feed	FF
\n	New line	LF
\r	carriage return	CR
\t	(horizontal) tabular	HT
\v	vertical tabular	VT
\\	Backslash	
\’	single quote	
\”	double quote	
\[	left square bracket	
\]	right square bracket	

Test-Programm

```
1  --~ print ( "“a", "“a" )
2  --~ print ( "“b", "“b" )
3  --~ print ( "“f", "“f" )
4  --~ print ( "“n", "“n" )
5  --~ print ( "“r", "“r" )
6  --~ print ( "“”", "“”" )
7  --~ print ( "“v", "“v" )
8  print ( "\"", "“" )
9  print ( "\'", "\'" )
10 print ( "\"", "“" )
11 print ( "\[", "\[" )
12 print ( "\]", "\]" )
```

The Result:

“” ”

“’ ’

“” ”

“[[

“]]

[1]When you need an overview on ASCII, you can take the ASCII Chart. In the right lower corner you find the octal value (Source: http://dante.ctan.org/tex-archive/info/ascii.tex)

2 Operators

2.1 Basic Arithmetics

Negative Numbers get a leading - (Minus)

Binary operators:

Operator	Description	Example
+	Addition	a = b + c
-	Subtraction (negatives leading sign)	a = b - c
*	Multiplication	a = b * c
/	Division	a= b / c
^	Potentialisation	a = b^c

Test-Programm

```
1 print( '2 + 3 =', 2 + 3 )
2 print( '2 - 3 =', 2 - 3 )
3 print( '2 * 3 =', 2 * 3 )
4 print( '2 / 3 =', 2 / 3 )
5 print( '2 ^ 3 =', 2 ^ 3 )
```

The Result:

2 + 3 = 5
2 - 3 = -1
2 * 3 = 6
2 / 3 = 0.66666666666667
2 ^ 3 = 8

2.2 Comparison

== Equality

~= Imparity

< Lesser

<= Less equal

> Bigger

>= Bigger equal

Test-Programm

```
1 print( '2 == 3 ', 2 == 3 )
2 print( '2 ~= 3 ', 2 ~= 3 )
3 print( '2 > 3 ', 2 > 3 )
```

```
4 print( '2 < 3 ', 2 ¡ 3 )
5 print( '2 >= 3 ', 2 ¿= 3 )
6 print( '2 <= 3 ', 2 ¡= 3 )
```

The Result:
2 == 3 false
2 ~= 3 true
2 ¿ 3 false
2 ¡ 3 true
2 ¿= 3 false
2 ¡= 3 true

Test-Programm

```
1 print( "'A' == 'B'", 'A' == 'B' )
2 print( "'A' ~= 'B'", 'A' ~= 'B' )
3 print( "'A' > 'B'", 'A' ¿ 'B' )
4 print( "'A' < 'B'", 'A' ¡ 'B' )
5 print( "'A' >= 'B'", 'A' ¿= 'B' )
6 print( "'A' <= 'B'", 'A' ¡= 'B' )
```

The Result:
'A' == 'B' false
'A' ~= 'B' true
'A' ¿ 'B' false
'A' ¡ 'B' true
'A' ¿= 'B' false
'A' ¡= 'B' true

Test-Programm

```
1 print( "'a' == 'A'", 'a' == 'A' )
2 print( "'a' ~= 'A'", 'a' ~= 'A' )
3 print( "'a' > 'A'", 'a' ¿ 'A' )
4 print( "'a' < 'A'", 'a' ¡ 'A' )
5 print( "'a' >= 'A'", 'a' ¿= 'A' )
6 print( "'a' <= 'A'", 'a' ¡= 'A' )
```

The Result:
'a' == 'A' false
'a' ~= 'A' true
'a' ¿ 'A' true
'a' ¡ 'A' false
'a' ¿= 'A' true
'a' ¡= 'A' false

2.3 Logical Operators

not Logical No

and Logical and

or logical or

Test-Programm

```
1 print( "true and false =", true and false )
2 print( "true or false =", true or false )
```

The Result:

true and false = false
true or false = true

2.4 Braces in logical Expressions

Test-Programm

```
1 print( " true or  false  and false  =", true or  false  and false
  )
2 print( " true or (false  and false) =", true or (false  and false) )
3 print( "(true or  false) and false  =", (true or  false) and false
  )
```

The Result:

true or false and false = true
true or (false and false) = true
(true or false) and false = false

2.5 Braces in logical Expressions with Negation (not)

Test-Programm

```
1 print( "not true and false       =", not true and false )
2 print( "not (true and false)     =", not (true and false) )
3 print( "true and not false       =", true and not false )
4 print( "not true or false        =", not true or false )
5 print( "not (true or false)      =", not (true or false) )
6 print( "true or not false        =", true or not false )
```

The Result:

not true and false = false
not (true and false) = true

true and not false = true
not true or false = false
not (true or false) = false
true or not false = true

3 Variables in Lua

- You must not declare variables, they are created when needed.
- The type of a variable is not fix, it is defeined by the actual value.
- Variables are global, if they are not declared *local*.
- Values can be thefollowing types:
  - Nil
  - Numbers
  - Literals (Characters, Words...)
  - Boolean (true/false)
  - Table
  - Functionen (Special case of a block)

Test-Programm

```
1 print( var )
2 var = "Hello World"
3 print( var )
```

The Result:
nil
Hello World

- Undefined variables are *nil*
- Assignment is done with =

Test-Programm

```
1 print( var1, var2 )
2 var1, var2 = 1, 4
3 print( var1, var2 )
```

The Result:
nil nil
1 4

- Multiple assignment is possible

4 Branches with If

4.1 If-Branch with Else

Test-Programm

```
a = 1
if a == 1 then
        print( 'a ist eins' )
else
        print( 'a ist nicht eins sondern ', a)
end
```

The Result:
a ist eins

Test-Programm

```
a = 999
if a == 1 then
        print( 'a ist eins' )
else
        print( 'a ist nicht eins sondern ', a)
end
```

The Result:
a ist nicht eins sondern 999

4.1.1 If with multiple Ifs

Test-Programm

```
a = 2
if a == 1       then
        print( 'a ist eins' )
elseif a == 2 then
        print( 'a ist zwei' )
else
  print( 'a ist nicht eins oder zwei')
  print( 'a ist  ', a)
end
```

The Result:
a ist zwei

4.2 Case-Statement

There is no Case-statement, it must be made with if-elseif-statements.

5 Loops

5.1 For-Loop

Parameter for the *For*-command:

1. Initial value
2. End value
3. Inkrement

Test-Programm

```
1 for variable = 0, 10, 2 do
2         print ( variable )
3 end
```

The Result:

0
2
4
6
8
10

Test-Programm

```
1 for variable = 0, 1, .5 do
2         print ( variable )
3 end
```

The Result:

0
0.5
1

The loop value doesn't need to be a in whole numbers.

Test-Programm

```
1 for variable = 0, 1, .5 do
2         print ( variable )
3 end
```

The Result:
0
0.5
1

Decrement works also.
There is a special variant of the for-command for tables.

5.2 While-Loop

Test-Programm

```
1 i = 1
2 while i ¡= 5 do
3             print (i)
4             i = i + 1
5 end
```

The Result:
1
2
3
4
5

- Loop with starting condition.
- If the starting condition is false, the block is never executet

5.3 Repeat-loop

Test-Programm

```
1 i = 1
2 repeat
3             print (i)
4             i = i + 1
5 until i ¿ 5
```

The Result:
1
2
3
4
5

- Loop with condition in the end.
- There is at least one execution of the block.

5.4 Abortion

You can stop the loop with *Break*.

Test-Programm

```
for variable = 1, -1, -.5 do
        if variable == 0 then
                print "Null erreicht"
                break
        end
        print ( variable )
end
```

The Result:
1
0.5
Null erreicht

6 Functions

Test-Programm

```
function Test()
        print( "Hallo Welt" )
end

Test()
```

The Result:
Hallo Welt

Return values are defined with *return*.

Test-Programm

```
function test()
        return  "Hallo", "Welt"
end

v1, v2 = test()
print( v1 )
print( v2 )
```

The Result:
Hallo
Welt

Multiple return values are possible.

Test-Programm

```
function test()
        return  "Hallo", "Welt"
end

v = test()
print( v )
```

The Result:
Hallo

If you return multiple values without enough receivers, the additinal return values are ignored.

Test-Programm

```
function test()
        return  "Hallo Welt"
end

print ( test() )
```

The Result:
Hallo Welt

6.1 Functions with Parameter

Test-Programm

```
function summe( v1, v2)
        return  ( v1 + v2 )
end

print( summe( 1, 2 ) )
print( summe( 2, 3 ) )
```

The Result:
3
5

- Parameter start with _ (it's a convention, not a must)

Test-Programm

```
a = 'vorher'
function summe( _v1, _v2)
        a = _v1 + _v2
        return  a
end

print( summe( 1, 2 ) )
print( a )
print( _v1 )
```

The Result:

3
3
nil

- Parameter are local.
- Other variable are defined global.

Test-Programm

```
a = 'vorher'
function summe( _v1, _v2)
        local a = _v1 + _v2
        return  a
end

print( summe( 1, 2 ) )
print( a )
print( _v1 )
```

The Result:

3
vorher
nil

- If new varibales are defined with *local*, then they are only defined inside the block.