



LUA DATA TYPES

Prepared by: S. S. Kochetkov

Supervisor: T. I. Krasikova




WHAT IS “LUA”?

- **Lua** (/ˈluːə/ **loo**-ə, from Portuguese: *lua* [ˈlu.(w)ɐ] meaning *moon*) is a lightweight multi-paradigm programming language designed primarily for embedded systems and clients. Lua is cross-platform since it is written in ANSI C, and has a relatively simple C API.

- **nil**
- **number**
- **string**
- **table**
- **function**
- **userdata**

```
a = nil
b = 1
c = 'abc'
t = { a,b,c }
f = print
u = some_C_struct
```



```
a = "a string"  
b = 'is the second line'
```



Strings enclosed in double quotes are interpreted C-like escape sequences, starting with the "\" character (backslash):

\ B (blank)

\ N (newline)

\ R (carriage return);

\ T (horizontal tab)

\\ (Backslash);

\ " (Double quote);

\ '(Single quote).



Note

The symbol in the line may also be represented by their code using the escape-sequence:

`\ Ddd,`

where ddd - a sequence of no more than three digits.



Also quotes to define string double square brackets can also be used:

```
local a = [[The company "Kronos"]]
```


Defining line by using double square brackets allows you to **ignore all the escape-sequence**, ie, the string is created entirely as described..:

```
local a = [[string
string1
string2
    string3
]] - "String
    string1
    string2
        string3 "
```

Note

When determining the line by using double square brackets accounted for tabs, and transport.

Double brackets can be nested. In order to prevent any confusion, the symbol "equal" (=) is inserted between the brackets:

```
local a = [= [string definition [[string]] in Lua] =]
```

```
- To the term: "the definition of the line [[string]] in the Lua"
```

5. **Function.** Functions in Lua can be stored in variables, passed as arguments to other functions, and returned as the function result.
6. **Table.** The table is a set of pairs of the "key" - "value", which are called **fields** or table **elements**. As the keys and values of table fields may be of any type, except nil. The tables do not have fixed size: at any time, they can add an arbitrary number of elements.
7. **Userdata (user data).** It is a special type of data. Values of this type can not be created or modified directly in Lua-script.

Userdata used **to represent new types** created in the calling program or script libraries written in C. For example, the Lua extension library for « CronosPRO » use this type to represent objects such as:

1. Data banks (Bank class);
 2. database (Base Class);
 3. recording (Record class) and so on. n.
5. **Thread.** It represents a stream of execution. These flows are in no way associated with the operating system and supported solely by means of Lua.

Lua does not provide explicitly specify the type of the variable. The variable type is set at the time of **assignment of variable values**. Any variable value may be assigned to any type (whether, what type of value it contained before).

```
a = 123 - a variable of type number  
a = "123" - now has a variable of type string  
a = true - is now a variable of type boolean  
a = {} - is now a variable of type table
```

note

Variables table, function, thread and userdata do not contain the actual data, and **store links** to related sites. When you assign, transfer to the function as an argument and return of function as a result of copying objects does not occur, are copied by reference to them.

```
a = {} - create a table. The variable a link placed on the table  
b = a - b a variable refers to the same table as a  
a [1] = 10 - element table with index 1 is set to 10  
MsgBox (b [1]) -> '10'  
b [1] = 20  
MsgBox (a [1]) -> '20'
```

The remaining data are immediate values.

```
a = 10  
b = a  
a = 20  
MsgBox (a) -> '20'  
MsgBox (b) -> '10'
```

AS IN LUA GET THE TYPE OF A VARIABLE?

Type of value stored in a variable, you can find out by using the standard function **type**. This function returns a string containing the name of the type («nil», «number», «string», «boolean», «table», «function», «thread», «userdata»).

```
t = type ( "a string") - t is equal to "string"  
t = type (123) - t is equal to "number"  
t = type (type) - t is equal to "function"  
t = type (true) - t is equal to "boolean"  
t = type (nil) - t is equal to "nil"  
t = type (CroApp.GetBank ()) - t is equal to "userdata"
```

AS IN LUA CONVERT THE VARIABLE TYPE?

Lua if necessary **automatically converts the number of the line and vice versa**. For example, if the string value is an operand in an arithmetic operation, it is converted to a number. Similarly, the numerical value, to meet at the place where the string will be converted to a string.

```
a = "10" + 2 - a 12 well
a = "10" + 2 - a is "10 + 2"
a = "-5.3e-10" "2" - a well -1.06e-09
a = "string" + 2 - Error! It is not possible to convert "string" in the number of
```


The value of any type can be explicitly converted to a string using the standard **tostring** function.

```
a = tostring (10) - a is "10"  
a = tostring (true) - a "true" on  
a = tostring (nil) - a well "nil"  
a = tostring ({[1] = "This field 1"}) - a well "table: 06DB1058"
```

From the above example it is clear that **the contents of the table function tostring not converted**. Perform this conversion, you can use **render** function.

```
a = render (10) - a is "10"
```

```
a = render (true) - a "true" on
```

```
a = render (nil) - a well "nil"
```

```
a = render ({[1] = "This field 1"}) - a is "{[1] =" 1 field is ""}
```

To **explicitly convert a value to the number** you can use the standard function **tonumber**. If the value is a string that can be converted into a number (or already is a number), the function returns the result of conversion, otherwise it returns nil.

```
a = tonumber ( "10" ) - a is "10"
```

```
a = tonumber ( "10" .. "5." ) - a is equal to 10.5
```

```
a = tonumber (true) - a well "nil"
```

```
a = tonumber (nil) - a well "nil"
```



THANKS FOR ATTENTION

