Jeroen P. Broks

A beginner's guide to:

Table of Contents

1. Introduction to Lua and this book

1.1 What is Lua?

Lua is a scripting language, which works completely in an interpreted environment. It has been coded in C and has been set up to be easily implemented with any program written in C. It's been used for many kinds of application. For example for creating addons for utilities, and most of all for games.

Lua was designed by Roberto Ierusalimschy, Waldemar Celes and Luiz Henrique de Figueiredo in a university in Rio de Janeiro in Brazil, and is completely free. It was named after the moon as “lua” means “moon” in Portuguese. Despite Lua itself being set up in C, you don't need any knowledge of C in order to use it, due to language quite often being implemented in tools in which you may need it.

Lua is very extremely symplistic in its syntax and general setup, making it very easy to learn and understand, even when you never touched a programming language before. Still Lua has a kind of low-level approach and is despite being interpreted quite fast. Since it's specifically designed to work in a C environment, it's very popular in the professional industry and knowing the basics of Lua can get you some extra tickets into the professional gaming industry.

I will go into the deep of how Lua works later in this book, but the famous “Hello World” program is only one line: “print('Hello World!')”.

Lua has also been designed to not bother about platform specific issues. So basically a Lua script should work in Windows, MacOS and Linux and basically any platform you can think of.

1.2 What do you need to install for this course.

When it comes to Game programming one of the most complete Lua engines to make anything is LÖVE (http://love2d.org) but I advise you not to start with LÖVE right away. Before we get onto a complete engine like that it's essential that you first understand how Lua itself works. For that you need a Lua program you can run from the CLI. This program can only perform the core features, but it's good enough for testing what you've learned. If CLI tools are too difficult for you to understand, you can also use the Lua playground (https://www.lua.org/cgi-bin/demo). You can just enter the Lua code and the result will appear as soon as you click “run”. For training this will do.

To Test the playground just type “print('Hello World!')” in the textarea and then click “Run”, and “Hello World!” should appear below. When you use the cli tool type “cat > hello.lua” in the terminal when you use Linux, Mac or BSD and “copy con hello.lua” on Windows and type

“print('Hello World!')” and press enter and press ctrl-D in Linux/Mac/BSD and ctrl-z on Windows then type “lua hello.lua” and if you see “Hello World!”, you can see you installed the CLI tool correctly.

When you really get serious into Lua scripting you will need an editor that can handle Lua Scripting. I use the Lua Devolopment Tools for Eclipse, but that can be a rather complex to understand. Atom, Geany, SublimeText, NotePad++ should all be able to allow you to work with Lua scripts.

1.3 How to use this book

I will try to take you step by step into the secrets of Lua. I will show you many scripts. It could be wise to copy these and try what they do.

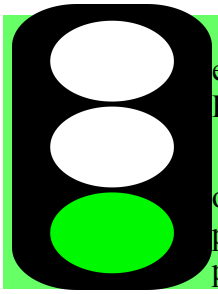
Code will look like this:

```
-- Test
a="Hello World"
print(a)
```

The colors you see is called “Syntax Highlight”. This will help you a lot keep some overview over your code.

When you see this icon and the text beside it with a blue background, it means that I got something very important to tell. Like all programming languages Lua can be full of trapdoors you can easily fall through, and you should therefore be aware of those. I make these warnings for a reason.

When you see this icon on an amber background, it means that I am about to tell you some typical nerdy stuff, that is just nice to know, but most of all aimed to people who can look beyond the regular stuff. If you have no understanding of what I tell here, then don't you worry one bit, as it's not really important to move on. Perhaps if you got more understanding of Lua and you want to read this over, hey, why not ;)



And lastly when you see this icon on a green background, I'll be giving you some exercise. The best way to learn to code, no matter if you do that in Lua, Pascal, BASIC or even in C is by doing it. And thus a few assignments never hurt.

These assignments may sometimes have some code snippets, but mostly you are on your own, but you area allowed to look things up in the previous chapter or paragraphs or sections to see the solution. Sometimes I may even give you some pointers of the expected output, so you can check a few things for yourself.

And with this all discussed, I wish you good luck with this book.

2. Hello World! Your first real program.

Most traditionally the first program you'll be writing in whatever language you're gonna learn is Hello World! It's been said that the first time Hello World was written was to demonstrate a prototype of the C programming language.

In C Hello World looks like this:

```
#include <stdio.h>
int main(void){
     printf("Hello World\n");
}
```

Not really much to it, even in C, although in C you need 4 lines and in Lua, just one.

```
print("Hello World")
```

Now “print” is a function. I'll go into the deep about what functions are later in this book, but for now it's important to know that a function can be used to make Lua do something. In the case of “print” that is to put something on the screen. “Hello World” is was we call a string. A string is a series of characters, usually used to form text. So in English we said to Lua *put the text “Hello World” on the screen.*

Let's test this out. Copy that line into a program or onto the playground and run it and see what happens. It should show “Hello World”.

When it comes to strings in nearly any language you will need quotes. In the example I used double quotes. Lua will also understand single quotes. It doesn't really matter if you use single or double quotes, as long as you are consequent in using the same quotes to end the quotes as you started them. I however advice double quotes. If you will ever consider moving on to C, Go, C# and some other languages in this family single quotes have a different meaning, and thus you can confuse yourself when you use single quotes (big exception is Pascal that requires single quotes).

Now I'm gonna take you into something important. Especially when you coded in BASIC or Pascal before you can really suffer here. Lua is case sensitive.

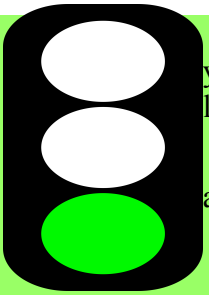
```
PRINT("Hello World")
```

This code will therefore not work. Try it and you will see Lua complain about a trying to call a “nil value” or something like that. I shall go into the deep about what a “nil value” is later, but for now let's suffice to say that a nil value means that you are trying to do something with something that does not exist, and therefore Lua can't handle that. Lua does make a difference between upper and lower case letters. So “print” and “PRINT” are therefore not the same. Especially when you move on to more complex Lua scripting you should be very strict on yourself on how you use upper and lower case or your scripts can become a big mess.

Now with this knowledge you can take this a little bit further. “print” will not only show text on screen. It will also move to the next line. And thus we can also do this:

So without trying this out in the playground or a cli tool, you must be able to tell what this program does:

```
print("USA stands for United States of America")
print("Its first president was George Washington")
print("It borders to Canada and Mexico")
```



Now write a program that shows your name and your age, your date of birth and your city of birth onto the screen. All data fields I asked for should have their own line in the output.

If you can do this, you have understood the basic of the Hello World sequence and then we can get on the move for the next chapter.

3. Variables

Like any programming language Lua handles data most of all through variables.

Unlike compiler based languages such as C, Go and Pascal, Lua does not require any variable declarations. A variable is technically created on the moment it's first asked for.

Variables can be defined, and read out. Read data can be put on the screen, or be used for creating new data, or be checked.

In this particular chapter I'll limit stuff to definitions of variables, reading and some nice manipulations.

Strictly speaking Lua only has 5 data types for variables: nil, boolean, string, number, table and function. I will not discuss function in this chapter. I will get to that once we'll really go into the deep of functions, and for tables same story.

Variables can also be global and local. I will get to the difference between the local and global in a later section. For now we'll only use globals. Global variables are available in your entire script, and that is all you need to know for now.

Definining variables is as easy as this:

```
a = 1 -- number
b = true -- boolean
c = "Hello World" -- string
d = { 3, 4, 5, 10 } -- table
e = function() print("Yo!") end -- function
n = nil -- nil
```

Now "--" in Lua means a comment. Lua will ignore everything that comes after that.

3.1 The nil type

Nil is a word that you'll soon hate when you get into Lua coding. It will haunt you wherever you go. Any variable you did not yet define will automatically be of the nil type, and the nil type only has one "value". Nil.

Nil just means your variable contains no data at all. In most cases nil cannot be used and thus an error will pop up if you try.

Nil can also be enforced like you can see in the code in the intro section of this chapter by simply saying "myvar = nil". Especially when working with tables this can get you far as this may cause Lua to automatically clean up all the memory taken by the table. More about that when I actually come to tables.

3.2 The number type

The number type is like the name suggests for storing numbers. These can be integers as well as floats. Now number variables are pretty important as they allow you to perform mathematical calculations, and even the simplest of programs (well except maybe Hello World) will often require you to do some calculations. Scoring points, deducting hitpoints, but also determining coordinates of the player or the enemies. The number type makes it all possible.

3.2.1 Definition and showing numbers

```
a = 1
print(a)
```

Yeah, well it's really that simple. This program will define 1 into variable a. And print will then show the content of a.

Note that now that we are using a variable and not a string we do not quote it. Due to that Lua understands this is a variable and will thus just show the value of variable a

```
a = 1
b = 5
print(b)
print(a)
```

Now 1 will be put in a and 5 will be put in b. Note that I first asked to show variable b and then variable a. This will cause Lua to first output 5 and then 1. It is very important that you put the commands in the correct order to prevent bugs. In this case it didn't matter in which order I defined the variables, but the order of the print instructions do.

3.2.2 Mathematics with variables

Now we get into actual programming.

For this I'll need to discuss operators. For mathematic formulas Lua has 6 operators at your disposal.

Addition	+
Subtraction	-
Multiplication	*
Division	/
Modulo	%
Empower	^

Lua does take in calculations the official order into account when multiple operators are used, so first empowering, then multiplication and division and lastly additions and subtractions, but just like in normal mathematics you can alter this order with parents ( ).

Now I'm sure this all sounds a bit like abracedabra. So let's explain by means of some examples.

When doing math in Lua you can use both numbers and variables in your formulas and you can use them in both your definitions as in function arguments, so you can do this:

```
print( 2 + 5 )
```

And you can also do this:

```
a = 2
print(a + 5)
```

Both examples will put 7 on the screen.

But this works too:

```
a = 2 + 5
print(a)
```

And yeah, you can also go this way:

```
a = 2
b = 5
c = a+b
print(c)
```

Anything is possible.

Now that's interesting, but can you also make a variable increment itself? Sure!

```
a = 1
a = a + 1
print(a) -- outputs 1
a = a + a
print(a) -- outputs 4
```

And basically you can go all the way with this with the other operators.

The “%” operator or “modulo” (in some programming languages this is the “mod” keyword) is an operator that can confuse some people. It contains the remainder value of a division.

8 % 4 = 0

9 % 4 = 1 (8:4=2 and 1 remains).

If you have been using C/C++/Go/C# before you will not like to hear this, but Lua has no shortcut incrementors or manipulation support. So a++ or a-- do not exist in Lua and there is also no variant for a+=4 or anything like that. I'm afraid this is something you'll have to deal with.

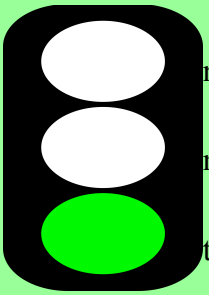
I've not been informed by any plans of support for this in the future. I wouldn't be surprised if it will be implemented, but for the good of this book, I'll do without them. :(

Now like I said before, Lua can handle more complex formulas by working with parents, in which like in real-life mathematics the stuff between parents takes priority

```
a = 4
b = 6
c = 2
d = (a+b)/c
print(d) -- outputs 5
```

Since a is 4 and b is 6 and c is 2 the formula send to d is basically *(4+6)/2*, so first calculate what is between the parents, well 4+6=10, so that makes 10:2 which is 5 and thus the output.

3.2.3 Assignment



Write a program in which you have variable a and b are defined with random numbers of your choice, as long as it's anything higher than zero.

Then let it output the result of adding the two variables, then subtract then multiplied by each other and then divided.

Then make the same program make the two variables double themselves and do the same sequence of mathematic results again.

Please make sure you read the assignment well and that you've understood it well, as this may be one of the more complex ones to understand at once.

3.3 The string type

3.3.1 An introduction to strings

Strings are in Lua pretty easy to use. Like I said strings are a chain of characters, usually used for text. Technically strings can have any length and you can do various things with them.

Defining a string variable is as simple as this:

```
mystring = "Hello World"
```

It's that easy. Now you can just use functions like print to put the content onto the screen.

```
mystring = "Hello World"
print(mystring)
```

Copying a string to another variable does not require special functions (like strcpy in C), but can just be done as easily as this:

```
mystring = "Hello World"
mystring2 = mystring
print(mystring2)
```

3.3.2 Concatenation

Now in Lua you cannot do any mathematic things on strings (seems logical, but there are some languages (like JavaScript) that can under certain circumstances, so that's why I explicitly note it), but there are ways to do some manipulations on strings.

One of the most common actions you can do with strings is concatenation, which is, simply appending a string to a string. Maybe that didn't make much sense, so let's demonstrate.

```
h = "Hello"
w = "World"
hw = h..w
print(hw)
```

So we placed “Hello” in h and “World” in w. In the third line we concatenate these strings and the result is stored in hw.

Now when you copied this program to the playground or a cli tool you will see something went “wrong”. The output is “HelloWorld” and not “Hello World”. This is not Lua's fault, but mine. I didn't put a space in the original variables. Spaces are just characters just like letters, so if you want them you must put them in, and if you don't want them, leave them out. Unwanted spaces or spaces not appearing when they should can easily happen in careless handing of strings, and sometimes the results are pretty downhearting and heart to find when debugging.

Now the funny part to concatenation is that Lua even allows numbers to be concatenated into strings.

```
s = "I am "..40.." years old"
print(s)

s = "I was born in "
y = 1975
s2 = s..y
print(s2)
```

Both examples work just fine.

3.3.3 C formatted strings in Lua

Now the C junkies reading this will very likely prefer C-formatting for concatenation, and therefore it's good to know that Lua has full support for this. There are basically 2 ways to do this, and it just comes down to what you think is best.

There is no variant to printf, but rather to sprintf and the result is just returned as a string. Two examples to do this:

```
name = "Jeroen"
year = 1975
mystring = string.format("My name is %s and I was born in %d",name,year)
print(mystring)
```

This is the simple way to do this. Of course to a real C programmer it goes without saying that you can use functions as parameters, so you can also do print(string.format(blah blah)).

There is also a kind of OOP way to do this and that'll look like this:

```
name = "Jeroen"
year = 1975
mystring = ("My name is %s and I was born in %d"):format(name,year)
print(mystring)
```

Now this way to go is also completely valid. Please note the (and the) in which the format string is set *is* required, or Lua will throw an error.

s

d