Unleash Your Coding Potential with Lua: The Versatile Scripting Language That Ignites Innovation!

Learning Lua

A Guide For Getting Started

Brian G. Burton, Ed. D.

Learning Lua
A Guide for Getting Started

Brian G. Burton, Ed.D.

Learning Lua: A Guide for Getting Started
By Brian G. Burton, Ed.D.

Printed in the Abilene, Texas, United States of America

Published by Burtons Media Group. See http://www.BurtonsMediaGroup.com/books for more information.

ISBN (eTextbook): 978-1-937336-10-3
(print): 979-8-857767-29-0

Version 1.0.0803.02 (August 2023)

Table of Contents

About the Author

Brian Gene Burton, Ed.D. is a professor, author, and game developer. He has written several textbooks, including “Learning Mobile App Development with Corona”, “Creating Multiplayer Games with Unity 3D”, and has contributed to several academic books on serious games and learning in virtual worlds. Dr. Burton teaches game development and virtual production and has an active YouTube station (https://www.youtube.com/@profburton). Dr. Burton presents and publishes his research internationally and enjoys sharing what he has learned.

Dedication

I dedicate this book to my loving wife whose support and encouragement kept me focused and writing. Thank you for keeping me focused and not running off on rabbit trails!

Preface

Who This Book Is For

While my focus and impetus for writing this book is that it be used as a textbook, I have also written it with the understanding that many (hopefully) are just interested in learning more about the Lua scripting language. As I wrote this book, it was with the expectation that this is your first time programming or you are not an experienced programmer.

How This Book Is Organized

We have broken the book into three parts, focusing each part on beginning concepts, intermediate, and advanced.

Conventions Used In This Book

Throughout the book, we have used a box to enclose script examples.

```
print (“Hello World”)
```

Using Code Examples and Fair Use Laws

This book was written to help you learn to develop applications and games with the Lua scripting language. In general, you may use the code in this book in your programs and documentation. You do not need to contact us for permission to reproduce a portion of the code. You don’t need to ask permission to write an app that uses any of the example code.

How to Contact Us

Please address any comments or questions to sales@BurtonsMediaGroup.com.
Join our discord server at https://discord.gg/tKSWMGh9fU

Why I Chose to Indie-Publish

The decision to Indie-publish was reached after a great deal of consideration. While there were numerous publishers interested (both academic and technical), I decided to release this first edition without the use of traditional publishers. There are many reasons why I made this decision, even though it will most likely lead to fewer sells.

First among my concerns was the price of the final book. I am sick of seeing textbooks at $100+. I feel such pricing places an undue burden upon students and schools. While publishers have cut the price slightly with the advent of eBooks and eTextbooks, it hasn't been enough in my opinion. By indie-publishing, I am only at the mercy of Apple, Amazon, Google, and Kobo.

My second concern was how rapidly software environments change. I personally hate having to purchase a new book for each major revision of the software. I have stacks of books that are now completely useless. I decided to publish this as an eTextbook, which allows me to update and provide it to you, the reader, more

rapidly. I will provide the updates between editions to the eBook to everyone who purchases the eTextbook through my website: http://www.BurtonsMediaGroup.com/books/book-update/

However, if you received a copy of this book, either through a torrent or a friend, please purchase your personal copy through my website. This will provide you with the most recent version of the textbook and encourage me to continue to keep it updated. While I am doing this to help my students, I have bills to pay, and my wife is amazingly creative at keeping my 'honey-do' list up-to-date. Help me to avoid that list by buying a legitimate copy of this book.

On the downside of indie publishing, I do NOT have a team of people to proof and double-check everything in this book. I am sure that typos were entered by gremlins during the night. To make things more challenging, I have dyslexia. I did hire a person to proof the final version of the book but having read many books that were published by major companies and found errors in their books, I am sure that errors remain in this one. Please let me know if you find a typo via email drburton@burtonsmediagroup.com.

Curriculum

Part I: Introduction and Setup

- History of Lua
- Installing Lua
- Hello World
- Comments
- Variables
- String Variables
- Mathematics and Math Library
- Functions
- Decisions
- Loops

Part II: Intermediate Lua

- Input/Output Library
- Tables
- Pairs & iPairs
- Closure
- OS Library
- Modules
- Recursion

Part III: Advanced Lua

- Objects
- Metatables and Metamethods
- Additional Resources

Part 1:
Introductory Concepts of Lua

Chapter 1: Introduction to Lua

Lua is a lightweight scripting language that is widely used for embedded systems scripting top applications and games. Over the subsequent chapters, we will cover the basics of programming and how to create your own games or applications using Lua. In this chapter, we will cover topics such as:

- What is Lua and why you should learn it
- Getting started with Lua
- Using Lua.org and installing Visual Studio Code
- Basic commands in Lua programming language

Why Learn Lua?

Lua is an easy-to-learn scripting language that is widely used to support many applications in gaming and the technology industry. It is used in many embedded systems, top applications, and games.

In addition to game development, Lua is used in various other applications such as Adobe Photoshop Lightroom, Wireshark, VLC media player, and more. It is also used as a scripting language for web applications and server-side programming, where its lightweight design and efficiency come in handy.

What is a ‘lightweight’ scripting language? A lightweight programming language is one that has a small footprint and low overhead, meaning it requires fewer computer resources, such as memory and processing power, to run.

Lua is considered lightweight for two reasons:

- The Lua interpreter is very small and requires minimal resources to run. This makes it easy to embed into other applications, such as games or other software, without adding significant overhead.
- Lua is designed to be highly modular, meaning that it can be customized and extended to meet the needs of specific applications. This allows developers to create programs that are tailored to the needs of their application.

The most popular place you will find Lua in use is in games or game engines such as Roblox, Solar 2D, LÖVE 2D, and Defold:

1. Roblox: Lua is the primary scripting language used in Roblox. The Lua programming language is used to program the behavior of objects, create user interfaces, and more.
2. Solar 2D, formerly known as Corona SDK, is a cross-platform game engine that uses Lua as its scripting language. Lua is used to create game mechanics, handle user input, and create user interfaces. Solar 2D provides a range of Lua modules and libraries that developers can use to create games, including physics

engines, audio and video libraries, and more. You can find a lot of tutorials and books by the author on Solar 2D on YouTube and BurtonsMediagroup.com.

3. LÖVE 2D is an open-source framework for 2D game development. I first fell in love with this framework when I discovered LOVR, which is built on the LÖVE 2D engine and allows you to create VR applications using Lua with the extension LUVR.

4. Defold is another game engine that uses Lua as its primary scripting language. Lua scripts in Defold are used to create game mechanics, manage game assets, and handle user input. Defold's Lua engine is optimized for performance and includes a range of features, such as live coding, debugging, and profiling tools that make it easier to create high-quality games.

One of the first places I experienced Lua was playing World of Warcraft, where Lua scripting quickly found its way into my heart as a powerful tool that made the game even more enjoyable.

The History of the Lua

The Lua (pronounced LOO-ah, meaning “moon” in Portuguese) scripting language’s development began in 1993 by a group of researchers from the Pontifical Catholic University of Rio de Janeiro in Brazil led by Roberto Ierusalimschy, Waldemar Celes, and Luiz Henrique de Figueiredo. The team were members of

the Computer Graphics Technology Group (Tecgraf). Prior to this time, Brazil has stringent limitations on the importation of computer hardware and software, creating an atmosphere where Tecgraf and their clients were limited both politically and economically on what they could use for their projects.

In response to this challenge, the Tecgraf team began developing a new language that would be highly modular, extensible, and easy to learn. They drew inspiration from several existing programming languages, including Scheme, Modula, and Smalltalk, and incorporated many innovative features of their own.

The Lua language quickly gained popularity among developers in Brazil and around the world. Today, Lua is widely used in a variety of applications, from game development to scientific computing to web programming, and has become an important part of the programming landscape. The original goal was to create a language that was simple, efficient, and flexible enough to be used in a variety of applications.

The name "Lua" means "moon" in Portuguese, and it reflects the creators' fascination with astronomy. The language was initially designed to be embedded into other programs, providing an easy way for developers to extend the functionality of their software. Lua continues to be maintained by the LabLua, a part of the Computer Science department at the Pontifical Catholic University of Rio de Janeiro.

Lua Versions

The most current version of Lua is 5.4.6 at the time of this writing. Updates are released on a semi-regular basis as the language continues to evolve.

LuaJIT, or the Lua Just-In-Time interpreter, can take Lua script and convert it into machine code that can be executed directly by the computer's processor. This process of converting Lua code to machine code is referred to as "compilation."

By using a JIT compiler like LuaJIT, Lua programs can run much faster than they would with just the standard interpreter alone. This is because the JIT compiler can optimize the code in real-time, based on how it's actually being used during runtime.

In simple terms, LuaJIT takes Lua code and makes it run faster by compiling it into machine code on the fly. This makes Lua programs run more efficiently and quickly, which can be especially beneficial for performance-intensive applications like games and scientific simulations.

Getting Started with Lua

You can try Lua on the internet at https://www.lua.org/demo.html. While I do not recommend this as a long-term solution, it will allow you to try Lua and use follow along with the majority of examples in this textbook.

To use Lua as a developer, you should use it as a part of an IDE (Integrated Development Environment). There are several popular IDEs that provide Lua support. For the purposes of simplicity, we will be using Visual Studio Code from Microsoft which is fast, lightweight, and works the same on Mac and Windows. Some of the most popular tools that use Lua have their own IDE that allows you to easily incorporate your Lua code into their applications.

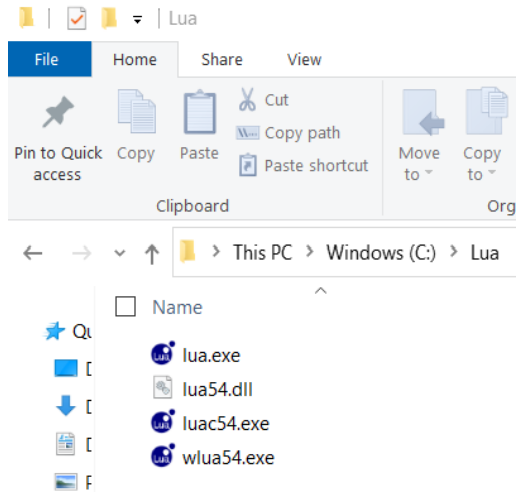
To install Lua on your personal computer, download the Lua binaries (i.e. the Lua Interpreter) from https://luabinaries.sourceforge.net/download.html.

LuaBinaries 5.4.2 - Release 1

lua-5.4.2_Sources.tar.gz	Source Code and Makefiles
lua-5.4.2_Sources.zip	Source Code and Makefiles
lua-5.4.2_Win32_bin.zip	Windows x86 Executables
lua-5.4.2_Win32_dllw6_lib.zip	Windows x86 DLL and Includes (MingW-w64 6 Built)
lua-5.4.2_Win64_bin.zip	Windows x64 Executables
lua-5.4.2_Win64_dllw6_lib.zip	Windows x64 DLL and Includes (MingW-w64 6 Built)
lua-5.4.2_MacOS1011_bin.tar.gz	MacOS X Intel Executables
lua-5.4.2_MacOS1011_lib.tar.gz	MacOS X Intel Library and Includes

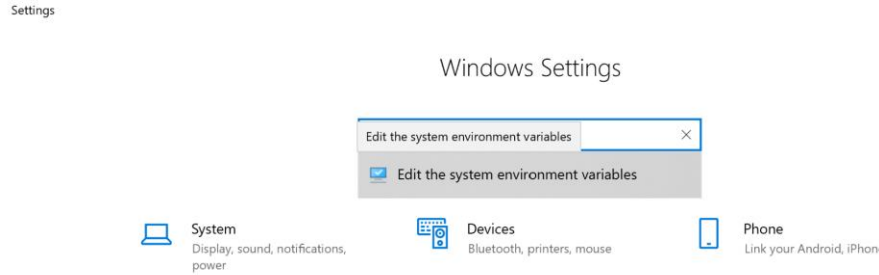
Extract the download for your operating system to a folder on your hard drive. I extracted Windows 64 (i.e. Windows 10 or Windows 11) to C:\Lua.

Rename the Lua54.exe to Lua.exe. This will simplify your life.



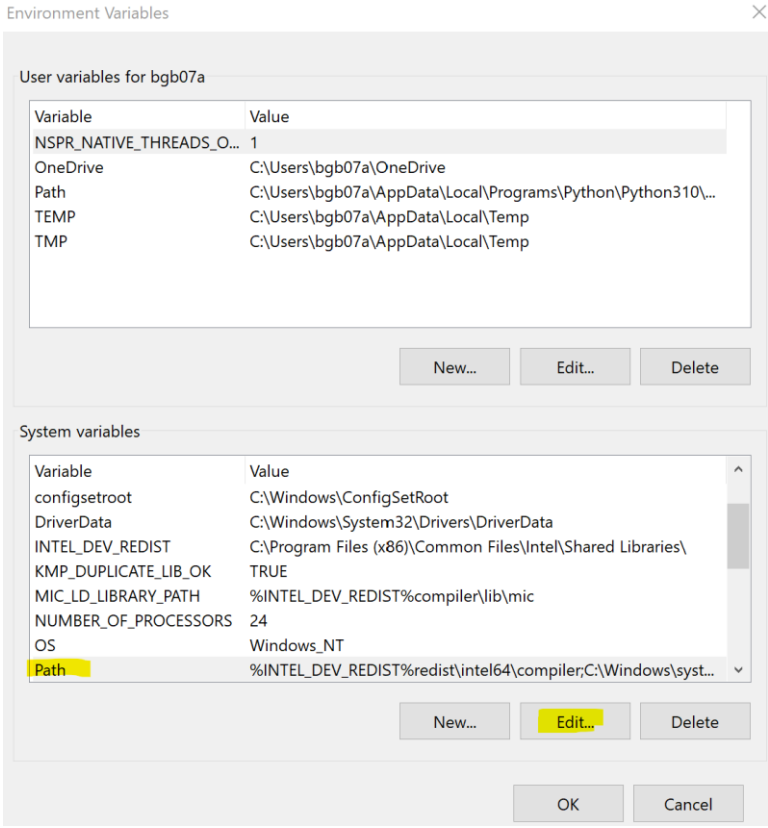
One more step to configure Lua for your development environment. In Windows, you will need to update the path to your Lua binary so that you can call the executable from a command line.

Open the System Environment Variables from your Settings.

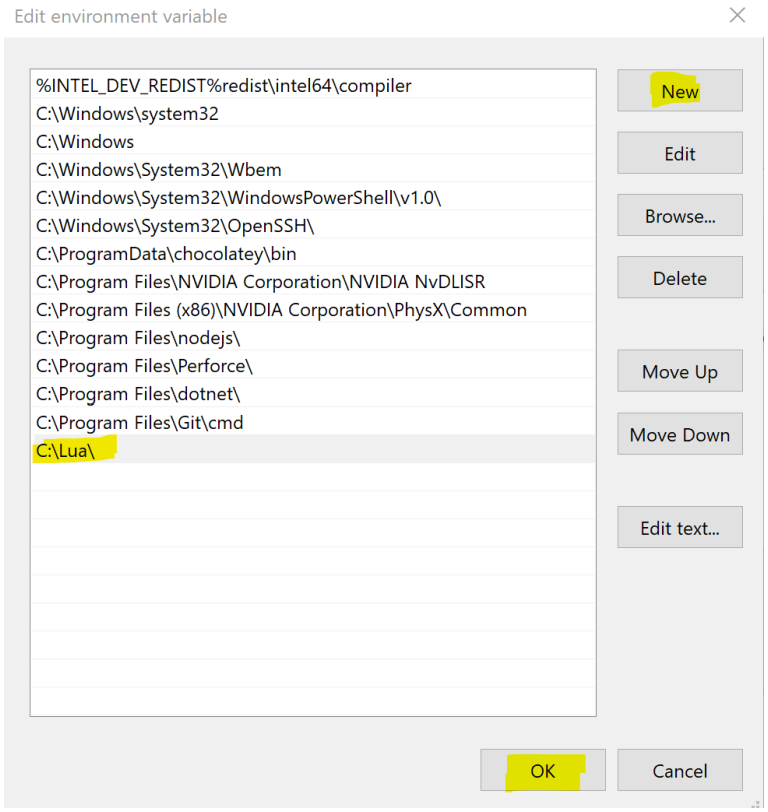


Click the “Environment Variables” at the bottom of the window.

In the “System variables” window, scroll down to “Path”, then click the Edit button.



Click the New button. On the new line, type the path to your Lua binaries. Just the path, you do not need to include the name of the executable file. Click OK when you are finished.



You have Lua ready to run on your system. Now we can install and configure Visual Studio Code.

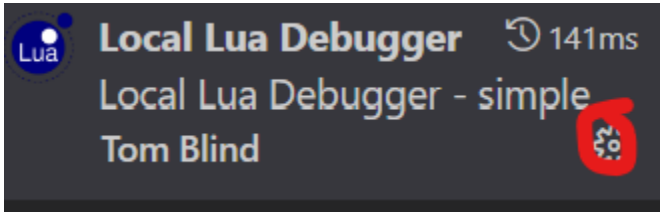
You can download Visual Studio Code from https://code.visualstudio.com/download. It is available for Linux, Mac, and Windows.

Follow the standard installation process for your platform. When you launch Code, you will need to install the Lua extension.

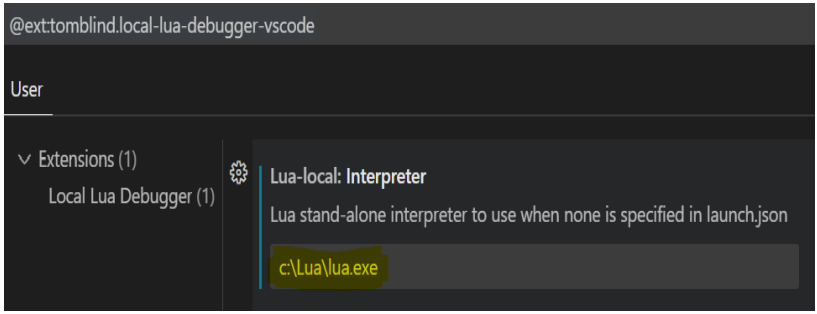
Once Visual Studio Code is installed, open it and click on the "Extensions" icon on the left-hand side of the window.

In the search bar, type "Lua" and hit enter. You should see a few results. The one you want to install is called "Lua Language Server" by sumneko. Click on the green "Install" button to install it.

Next, we will install the connection to the Lua interpreter. Scroll down to “Local Lua Debugger” by Tom Blind. Click on “install.” Finally, click on the Settings button



In the Extension Settings, type in the path to your Lua interpreter



Now we can begin scripting in Lua in Visual Studio Code!

Hello World in Lua

One of the basic commands in Lua programming language is the 'print' command. You can use the print command to output simple information. For example, to output "Hello World" in Lua, you can use the following command:

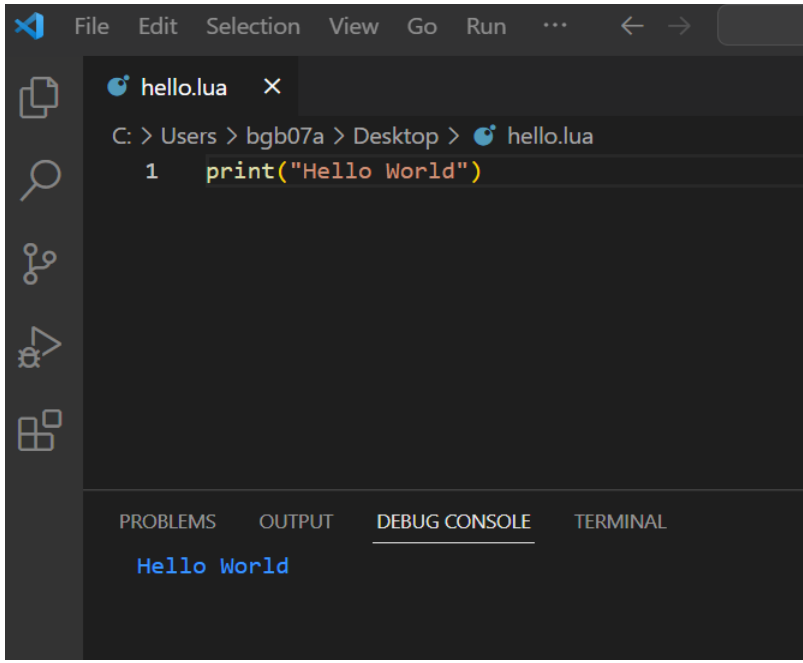
```
print("Hello World")
```

Save the file as “hello.lua”.

To see the program run, you will need a terminal window. Click on the three dots beside Run in the menu bar at the top of the window and select Terminal > New Terminal.

This will open a terminal window at the bottom of your screen. Tap the F5 key or click the Run > Start Debugging in the menu bar.

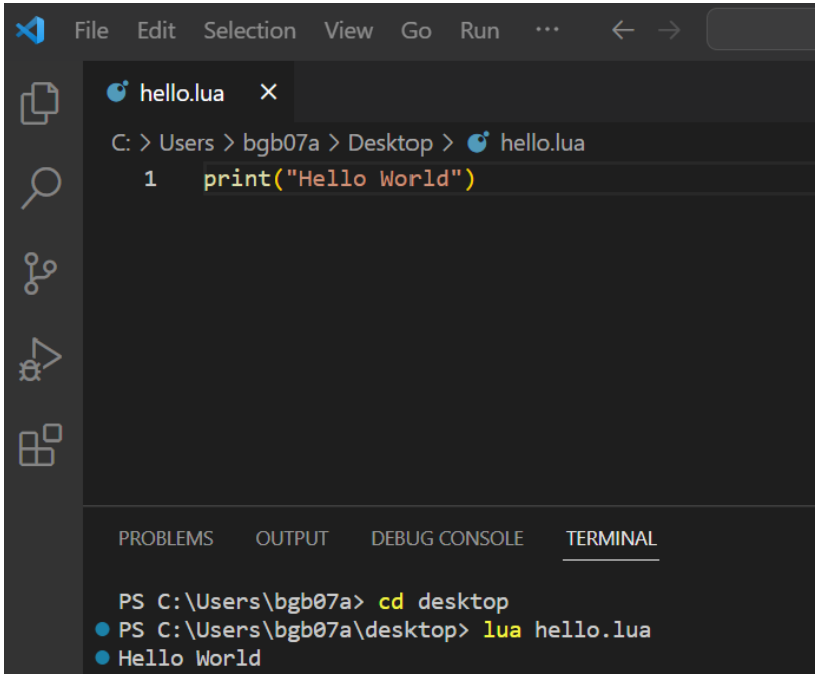
You should see “Hello World” displayed in the terminal window.



This is making use of the Local Lua Debugger that we installed through the extensions. Visual Studio Code is now able to execute any Lua program and display the results in the Debug Console or Output.

You can also run the program from the terminal window. Click on “Terminal” in the terminal window. Navigate to the folder where you saved hello.lua. In my case, I saved it on my desktop.

Type **lua hello.lua** and you should see the same results as pressing F5.



By including the path to the Lua folder where we save the binary files, we can call the lua executable from any command line on our system.

Troubleshooting in Lua

In Lua, the **print** statement is primarily used to output information to the console or terminal. This allows developers to quickly see the current value of variables, where they are at in the program, and to perform basic troubleshooting. Print is one of the most useful tools for being able to troubleshoot quickly and easily in Lua.

In more complex programs, it can be helpful to see what line or file you're currently working on. Using the print command, you can easily enter what line and file you're working on. This makes troubleshooting much easier, especially in large programs. Adding a comma between information in the print adds a tab break between the two parts, making it easier to read and get the information on the screen quickly and easily.

Another useful tool for troubleshooting in Lua is the ability to concatenate strings using two periods (“**..**”) . Concatenation is the process of joining two strings together. For example, you can join the two strings "hello" and "world" together by typing "print('hello'..'world')". Note that concatenation joins the two strings as they are, it does not add any spaces.

Documenting Your Program

Commenting your code is critical in programming, especially if you ever hope to do maintenance, troubleshooting, or work on it the next day. In Lua, comments can be added by typing two dashes or minus signs.

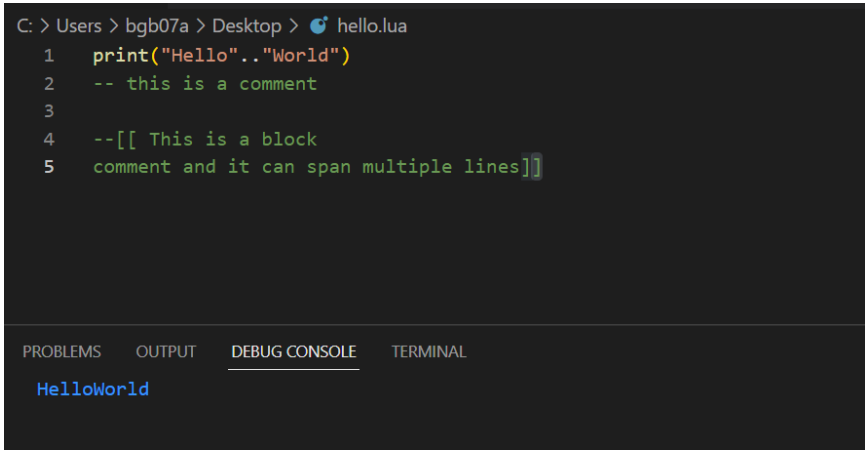
```
-- It is a good idea to comment your code
```

Anything after the minus signs will be treated as a comment and not evaluated by the interpreter; it ignores everything in a comment.

Inline comments can also be added by placing two dashes at the end of a line.

```
print (“Hello world!”) -- Everything after the dashes is a comment
```

Block comments can be added by typing two dashes followed by two bracket characters. The block comment is ended with two right brackets.



Conclusion

Lua is a very powerful scripting language that is easy to learn and widely used throughout the industry. In this chapter, we covered the basics of Lua programming language, including what it is, why you should learn it, how to get started, and basic commands. With this knowledge, you can start developing your own games or applications using Lua.

Questions

1. What is the primary purpose of the print command in Lua?
2. What is Lua and why is it widely used?
3. What are the topics covered in the chapter about Lua programming language?
4. What is the meaning of the term 'lightweight' scripting language?
5. What are the reasons why Lua is considered a lightweight scripting language?
6. What are some of the applications that use Lua besides gaming?
7. What are some of the popular game engines that use Lua?
8. Who developed the Lua programming language and why?
9. What is the current version of Lua and how often are updates released?
10. What is LuaJIT and how does it optimize Lua programs?

Going Deeper

You can learn more about the history of Lua at lua.org

Dr. Burton's videos introducing Lua:
Introduction to the Lua Scripting Language - https://youtu.be/-iU1pCgmjx4
Using Print and comments in Lua - https://youtu.be/7sj-qf99RPk

Chapter 2: Variables

Variables are a critical component of all programming languages. In this chapter, we will learn about the eight different types of variables available in the Lua scripting language.

Learning Objectives

In this chapter, we will learn about Lua's variable types. By the end of this chapter, you should be able to:

- Know the difference between local and global variables.
- Understand the default variable types in Lua.
- Explain how to use the "type" command in Lua.

Naming Variables

When we use a variable, we are creating an object. Just as in the real world, an object can be anything. And, of course, we need a name for the object. Whenever we assign an object to a variable, we are essentially giving that object a name. Everything that the object is will be stored and referred to through that assigned name. Just as you are called by a name, so are the objects within Lua. Using this name, we can use the object later in the app.

It is critical that each object has a unique name. Imagine a teacher asking Pat to answer a question in a class. But everyone in the class is named Pat. We

have the same problem if we reuse the same variable names. Make your objects feel special; give each one a unique name.

There are a few rules to the naming of variables:

- A variable can be any combination of letters, numbers, or underscores
- A variable must not begin with a number
- A variable cannot contain a space or any symbol except an underscore
- Variables are case-sensitive. myVariable is not the same as MyVariable

Valid	Invalid
Variable1 My_Variable _variable variable12345	1Variable My Variable -variable variable12.345

Local vs. Global Variables

Most of us have the experience, at one point or another, where we are given a 'temporary' name. Perhaps during school, there were two students with the same first name in a class, so they would have their last initial also used: thus two Heathers became HeatherA and HeatherB. Or maybe you were given a nickname in school or athletics. Usually, these different names were short-lived or only used in limited situations.

In programming, we have two types of variables: local and global. As we progress through the rest of this book (and any other programming language that you learn) think of the differences this way: a local variable is a short-lived name given to an object, much like that short-lived nickname in school. A global variable is a long-lasting name and can be used throughout a program, anywhere in a program.

How do you tell them apart? Easy. A local variable will always have the keyword **local** in front of it the first time it is used in an app. A global variable will never have, or be initialized with the word local. Many Lua developers place an underscore in front of global variables so that they can be easily identified.

Declaring a local variable:

```
local textObject
local myNewPicture
local backgroundImage
```

Declaring a global variable:

```
textObject
_myNewPicture
backgroundImage
```

The preference in programming is to always use local

variables whenever possible. Local variables use less memory and will help you avoid naming problems in more complex programs.

Variable Types in Lua

In Lua, we have eight default variable types: nil, number, string, boolean, table, function, userdata, and thread. It should be noted that Lua is a dynamically typed language, meaning you don't have to explicitly declare a variable type when creating your variables. You just simply declare your variable, and the Just-In-Time Compiler (LuaJIT) will automatically type the variable to what it thinks is the correct variable type.

Below are the basic variable types and the information that they can store.

The Nil Variable Type

Nil is the default variable type in Lua. All variables when they are created are, by default, nil until they are given a value. Nil is also used to clear variables or make them available for garbage collection to free up memory. If you are done using a variable inside a programming project, you can set it to nil to remove it from memory.

Examples of nil variables:

```
local myVariable = nil
local newVariable
```

The Number Data Type

The number data type is used to store all numeric values. It is a double-precision floating-point real number for storage that includes for all types, including what we would normally consider an integer or other data precision types for numeric values. Internally, Lua stores all numeric values as a 64-bit double-precision floating-point number.

Examples of numeric variables:

```
local myInteger = 1
local myDecimal = 3.214
local anotherInteger = 0
```

Numeric Operators

Lua provides all of the standard operators for working with mathematical equations that you will find in other programming languages:

Addition operator +

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

Subtraction operator -

```
print (10 - 5)
```

Division operator /

```
print ( 8 / 2 )
```

Multiplication operator *

```
print (5*3)
```

Exponent operator (the power of) ^

```
print (5^2)
```

Lua calculates values from left to right. When working with mathematical equations, you can enforce values to be calculated first by wrapping them in parentheses. When Lua finds a mathematical equation, it always calculates the innermost equation contained within parentheses first, then works its way to the outer nested equations. For example

```
print (2 + 2 * 2 + 2)
-- will return a different result than
print ((2 + 2) * (2 + 2))

print (3 + 3 * 3 * 3 + 3)
-- will return a different result than
print ((3 + 3 * 3) * (3 + 3))
-- will return a different result than
print (((3 + 3) * 3) * (3 + 3))
```

Dividing by Zero

In Lua, all mathematical expressions return a numeric value, with the exception of equations where a number is divided by zero. For instance, take the following expression:

```
print ( 5 / 0 )
```

In many languages, this expression would raise an error. However, in Lua, this expression would print the value ‘Inf’, meaning infinite. ‘Inf’ is not usable in a numeric equation and thus should be caught where possible in a value check. This should preferably be carried out before the equation by checking if the

divisor value is zero. A quick way that you can check the result of the equation is to use the following code:

```
result = 5/0
print ( result == 1/0 )
-- outputs true
```

Lua provides quite an extensive number of math functions. We'll be looking at these in Chapter 4.

The String Data Type

The string data type is a more complex variable type that stores a sequence of characters. You can create a string using either single or double quotes, or you can create a multi-line string using double-square brackets [[ ]]. Strings have their own styles as well as escape characters that are available to us through Lua. We will explore the string data type further in the next chapter.

Examples of string variables:

```
local myString1 = 'Hello world'
local anotherString = "Hello world with double quote"
local myString2 = [[You can have multiple
lines of text when you
use square brackets. ]]
```

The Boolean Data Type

The Boolean data type is used to store both true and false values. Booleans work a little bit differently in the Lua language. We are not limited to just true or false values to represent or output the meaning of a boolean. Nil would also return a false.

Examples of boolean variables:

```
local aTrueBoolean = true
local aFalseBoolean = false
```

Tables in Lua

Tables are a powerful and useful tool inside of the Lua scripting language. They store the data as an associative array, meaning that it can store strings next to integers next to reals next to booleans. It doesn't make any difference, and they can be indexed by any other variable type inside the system. To create an array, you use the curly brackets to delineate it. We will discuss tables in more detail in Chapter 9.

Examples of tables:

```
local myTable = { }  --an empty table
local myTable = { 'First Item', 2, 'Third item' }
```

We hope you have enjoyed this preview of ***Learning Lua***. You can purchase the book as a PDF from BurtonsMediaGroup.com
Paperback and Kindle version are available from Amazon

Video Tutorials - Learning Lua on YouTube
26 video tutorials on the Lua language by the author of this book

Thank you for joining us on this journey of Learning Lua!

Check out our newest books and resources at
BurtonsMediaGroup.com
YouTube.com/@profburton