THE LUA PROGRAMMING LANGUAGE BEGINNER'S GUIDE

In this article, we'll cover the basics of the Lua programming language, including:

- What it is
- Main components
- Key features
- Pros & cons
- Use cases
- Lua vs other languages
- Installing Lua in a Windows environment
- Getting it up and running

Let's get started.

What is Lua?

Lua is a robust, lightweight, and embeddable scripting language that supports multiple programming methods, including procedural, object-oriented, functional, and data-driven programming.

As the primary focus on Lua is for scripting, it is rarely used as a standalone programming language.

Instead, it is used as a scripting language that can be integrated (embedded) into other programs written in mainly C and C++. It also supports other programming languages via third-party plugins (NLua/KeraLua for .NET/C#).

Popular use cases for Lua include:

- As a popular component in video game and game engine development. For example, Warframe, World of Warcraft, and CRYENGINE all use Lua.
- As a programming language in many network programs, like CISCO Systems, Nmap, and ModSecurity.
- As a programming language in industrial programs such as Adobe Lightroom and MySQL Workbench.
- As a library that developers can integrate with their programs to enable scripting functionality.

Being a scripting language, Lua does not have its own main application. Instead, it acts exclusively as an embedded part of the host application.

(Explore the most popular programming languages.)

How Lua works

There are two main components of Lua:

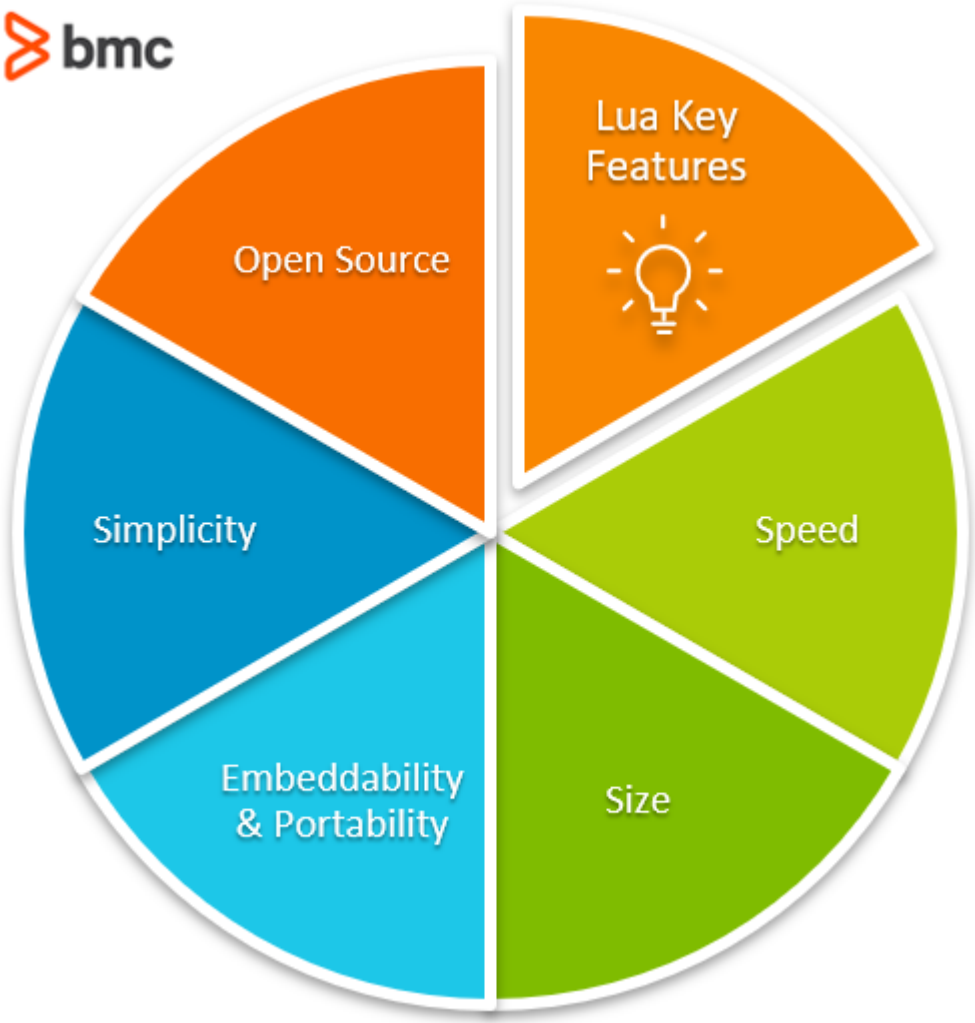
- The Lua interpreter
- The Lua virtual machine (VM)

Lua is not directly interpreted through a Lua file like other languages such as Python. Instead, it uses the Lua interpreter to compile a Lua file to bytecode. The Lua interpreter is written in ANSI C, making it highly portable and capable of running on a multitude of devices.

Usually, the compilation is done at the runtime. However, sometimes it can be done before the runtime to increase load times. Then the Lua virtual machine will run this compiled bytecode. The register-based architecture of the Lua virtual machine closely resembles actual hardware architectures, and it will increase the overall performance of the program.

Key features of Lua

So, what's so great about Lua? These are the defining features:



Speed

Lua is considered one of the fastest programming languages among interpreted scripting languages. In particular, Lua can perform large task orders faster than most other programming languages in both benchmarking and real-world scenarios.

For more speed, an independent implementation of Lua called LuaJIT uses a just-in-time compiler that makes Lua even faster.

Size

Lua has a considerably smaller footprint than other programming languages, with its complete source code and documentation taking a mere 1.3 MB. The Lua interpreter with all the standard libraries takes 278K, while the complete Lua library takes only 466K.

This small size is ideal when integrating Lua into multiple platforms, from embedded devices to massive game engines, where every byte is valuable.

Portability & embeddability

With its small size, the portability of Lua is nearly unlimited; any platform that supports the standard C compiler can run Lua out of the box. Lua's speed and size become huge advantages when embedding Lua with another programming language. That's because they can help increase the

speed of the program without hindering any existing functionality.

Importantly, Lua does not require complex rewrites to be compatible with other programming languages. Lua can be used with primary programming languages like C, C++, Java, C#, etc.. and other scripting languages like Perl and Ruby, further extending its usability.

Simplicity

Lua is simple in design yet provides powerful functionality. One of the core features of Lua is meta-mechanisms which enable developers to implement features—rather than providing a bunch of features directly in the language itself.

Lua also comes with incremental garbage collection, reducing memory usage and implementation complexity. Its sandboxing feature can be used to isolate functions and resources. This increases the security of the program and provides coroutines for multitasking.

All these features come with a simple syntax and easily understandable format so that anyone can easily pick up Lua and use it in their programs.

License

Lua is free and open-source software distributed under the MIT license. This means anyone can use Lua for any purpose without paying any licensing or royalty fees.

Advantages & drawbacks

Like any language, Lua has its pros and cons.

Advantages of Lua

- **Easy app integration.** Its high performance and small size make it easy to integrate Lua into applications.
- **Simple syntax.** Relatively simple syntax structure with around 20 dedicated keywords, which helps to dive into Lua programming easily.
- **Flexibility.** Without standard libraries, you can customize Lua to meet any need.
- **Cross-platform compatibility and support** for the standard C compiler allows Lua to run virtually anywhere.
- **Dynamic variables** in Lua allow defining variables without defining types, and the type is determined automatically at the runtime.
- **Easy debugging.** Simple and powerful debug library.
- **Plenty of documentation.** Comprehensive documentation to get Lua projects up and running quickly and the active community.

Disadvantages of Lua

- Limited error handling support can lead to longer debug times to identify the exact errors in a Lua script.
- All variables are created as global variables (global scope), which can lead to errors in variable assignments.

- Limited pattern matching support.

When to use Lua

As a scripting language without major limitations, you can use Lua for any scenario, from a simple backend script in a web server to complex game development.

Lua is highly prevalent in video game development as it can be used to create functionality without contaminating the overall performance while also keeping everything separate.

Another area that Lua excels is embedded programming, where size and performance are major concerns. Lua can be used in everyday applications to extend the existing functionality or create new features and functions.

Some popular games, programs, and services that use Lua are Dark Souls, Fable II, Garry's Mod, Wireshark, VLC, Apache, and Nginx Web Servers.

Lua vs other languages

How does Lua stack up against other languages?

Here's a look at the differences between the high-level general programming language Python, the high-level object-oriented Java, and web-focused Javascript—all compared to Lua.

bmc Comparing Common Programming Languages

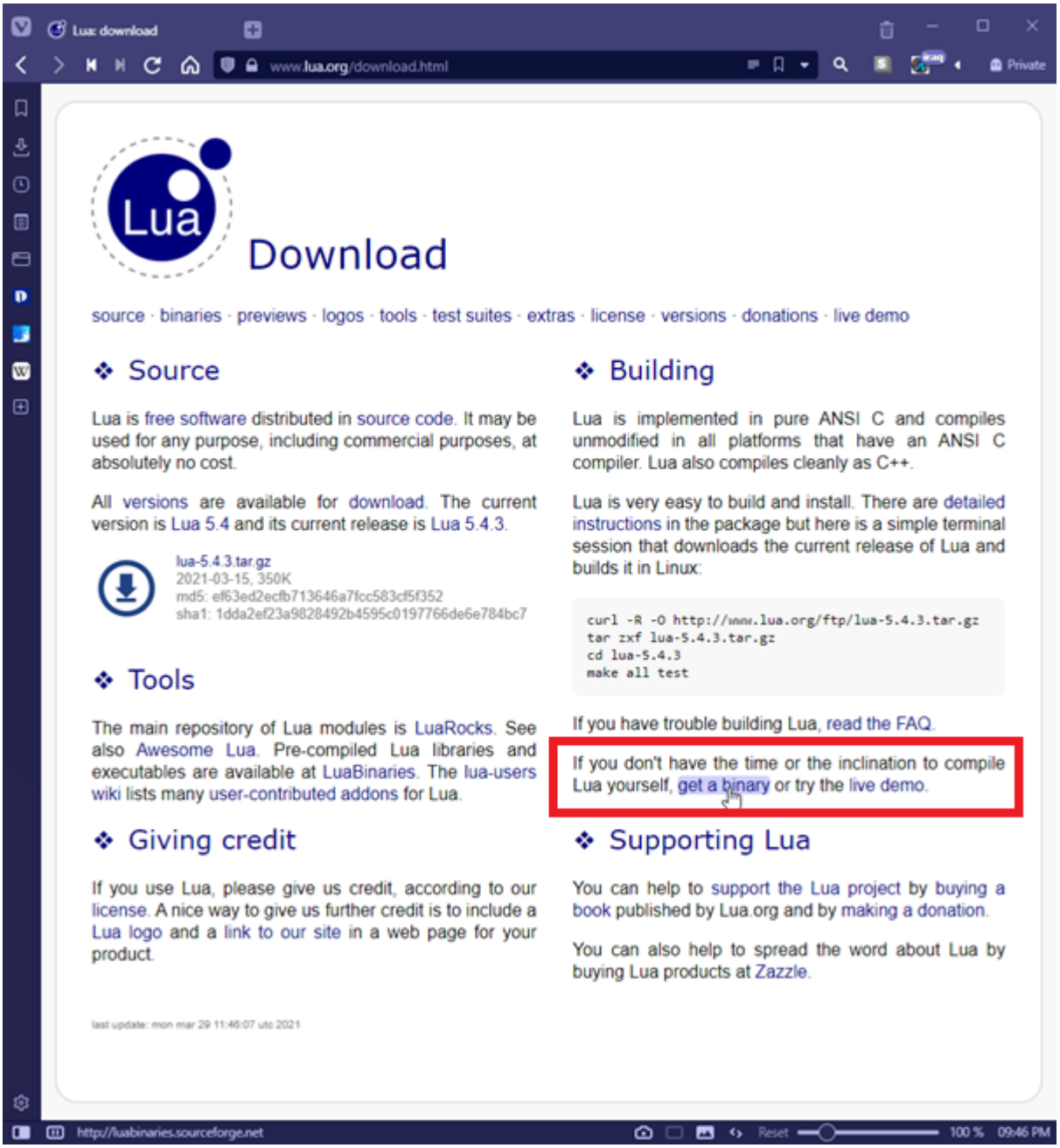
	Lua	Python	Java	JavaScript
Language Type	Scripting language	General purpose programming language	General purpose programming language	Web-focused programming language (Supports client- & server-side)
Usage	Embedded scripting	Multiple (From web development to data analytics)	Multiple (Desktop/mobile apps to Enterprise apps & services)	Web app development (Frontend & backend)
Compiled vs Interpreted	Compiled	Interpreted	Compiled	Interpreted
Package Management	Third-party tools (LuaRocks, LuaDist)	Pip	Maven & Gradle	NPM (Node Package Manager)
Programing Styles	• Procedural • Object-oriented • Functional • Data-driven	• Imperative • Functional • Object-oriented • Procedural	• Object-oriented	• Procedural • Object-oriented
Ease of use (Syntax)	Simple	Simple	Complex	Simple
Speed	Fastest	Relatively slow	Fast	Faster
Platform Support	Multi-platform	Multi-platform	Multi-platform	Multi-platform (Web oriented)

Installing Lua

Now, let's see how to set up a development environment in Windows. First, we'll install Lua.

Step 1

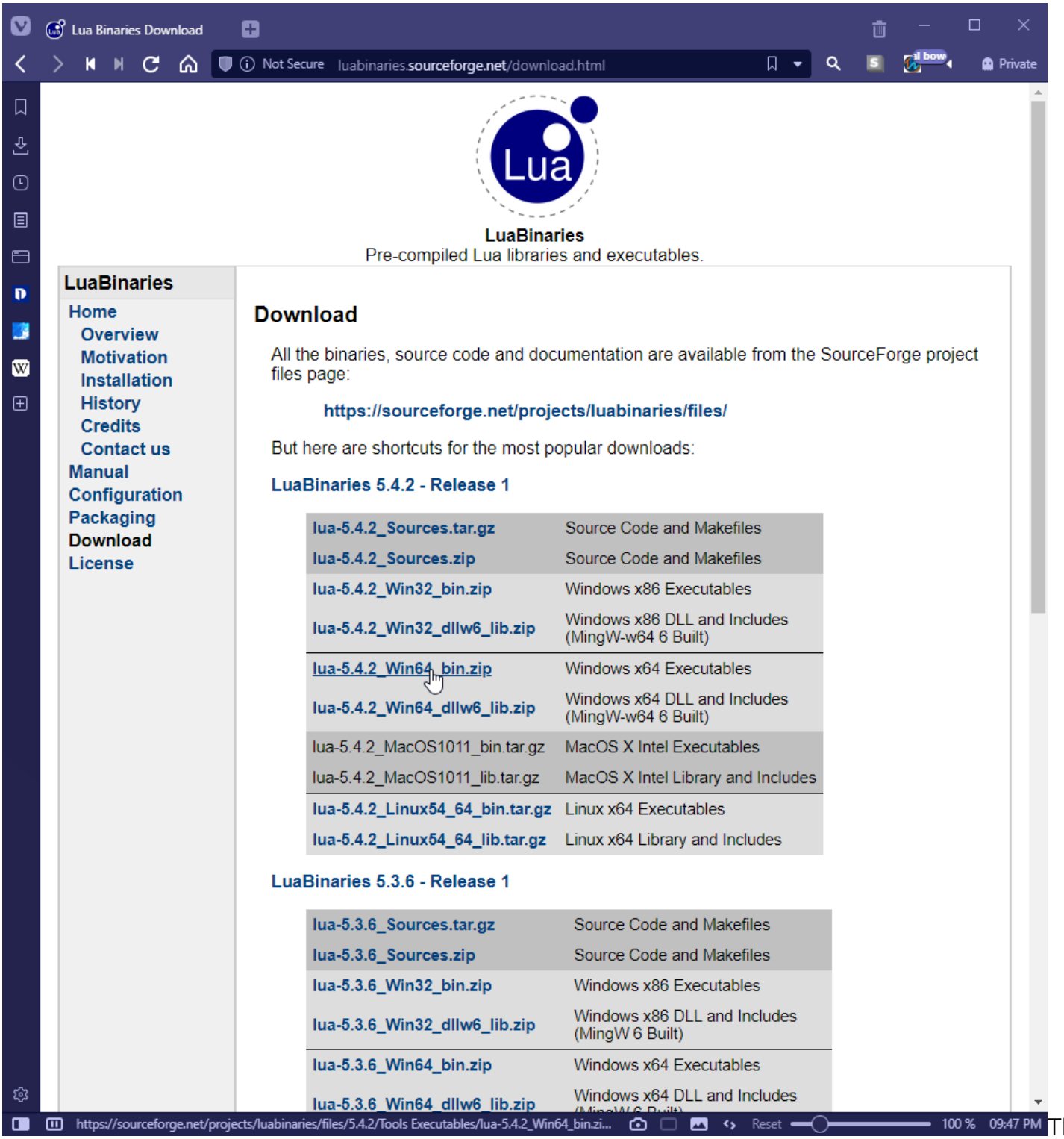
Navigate to the Lua.org download page. Here, we will be using a precompiled binary to install Lua in windows. So, click on "get a binary link" as shown in the below screenshot.



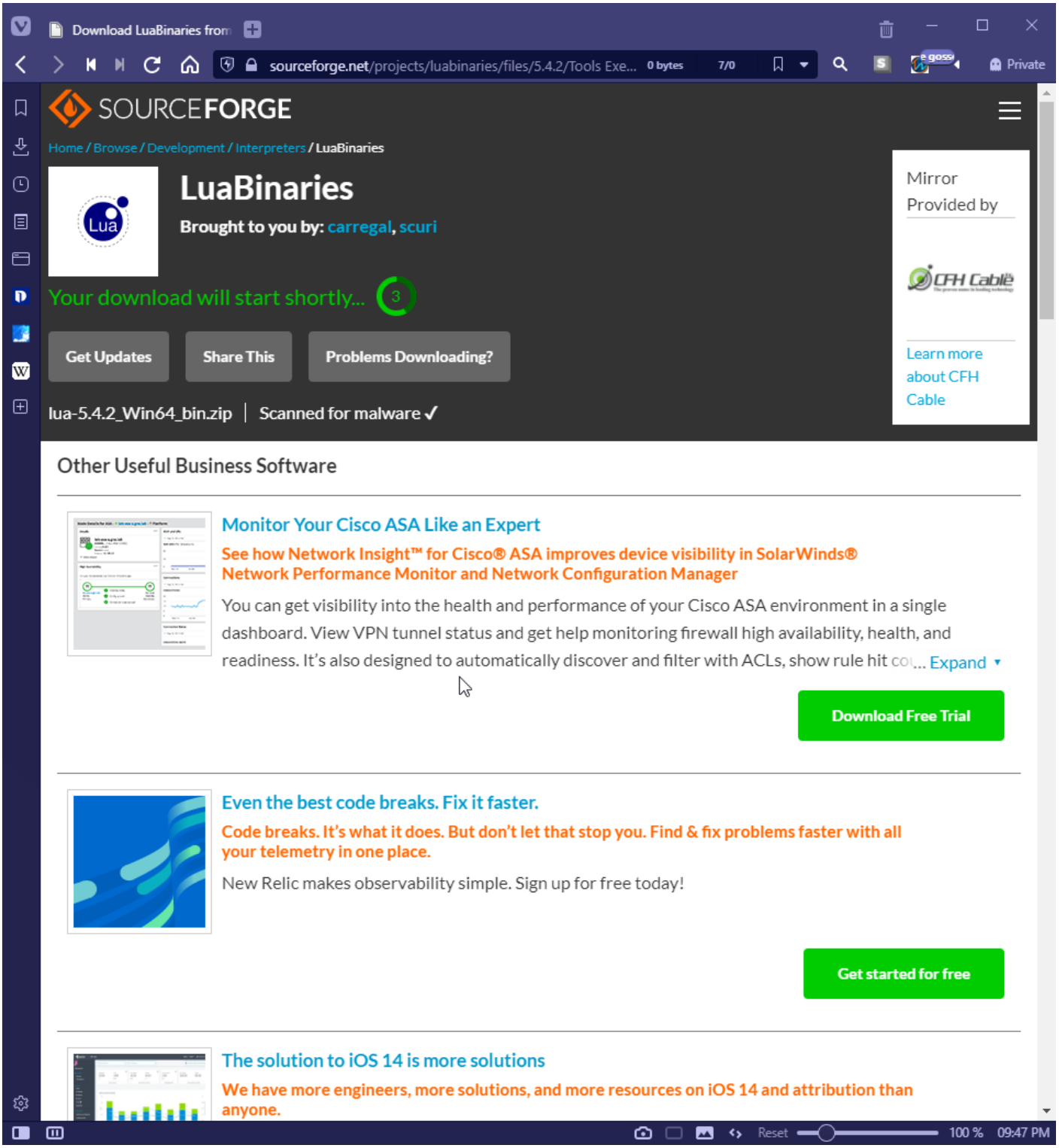
Step 2

Click "Download" on the LuaBinaries page, and you will be redirected to a page with a list of precompiled binaries. Select the appropriate version from that list.

We will be using the latest Lua version for Windows 64 bit.



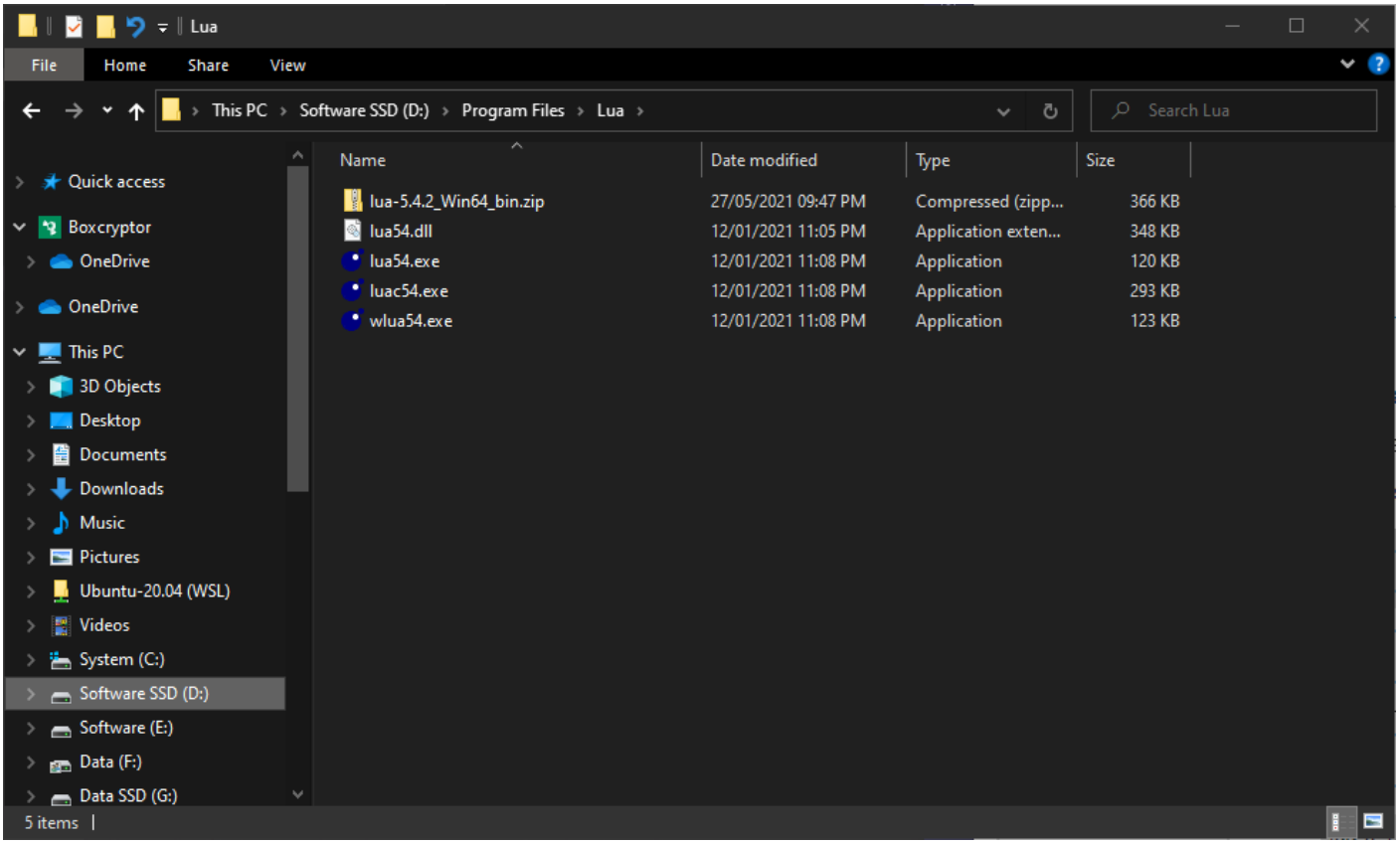
This will direct the user to a SourceForge page, where the binary will be downloaded.



Step 3

Move the downloaded Zip file to any location to store the binaries permanently.

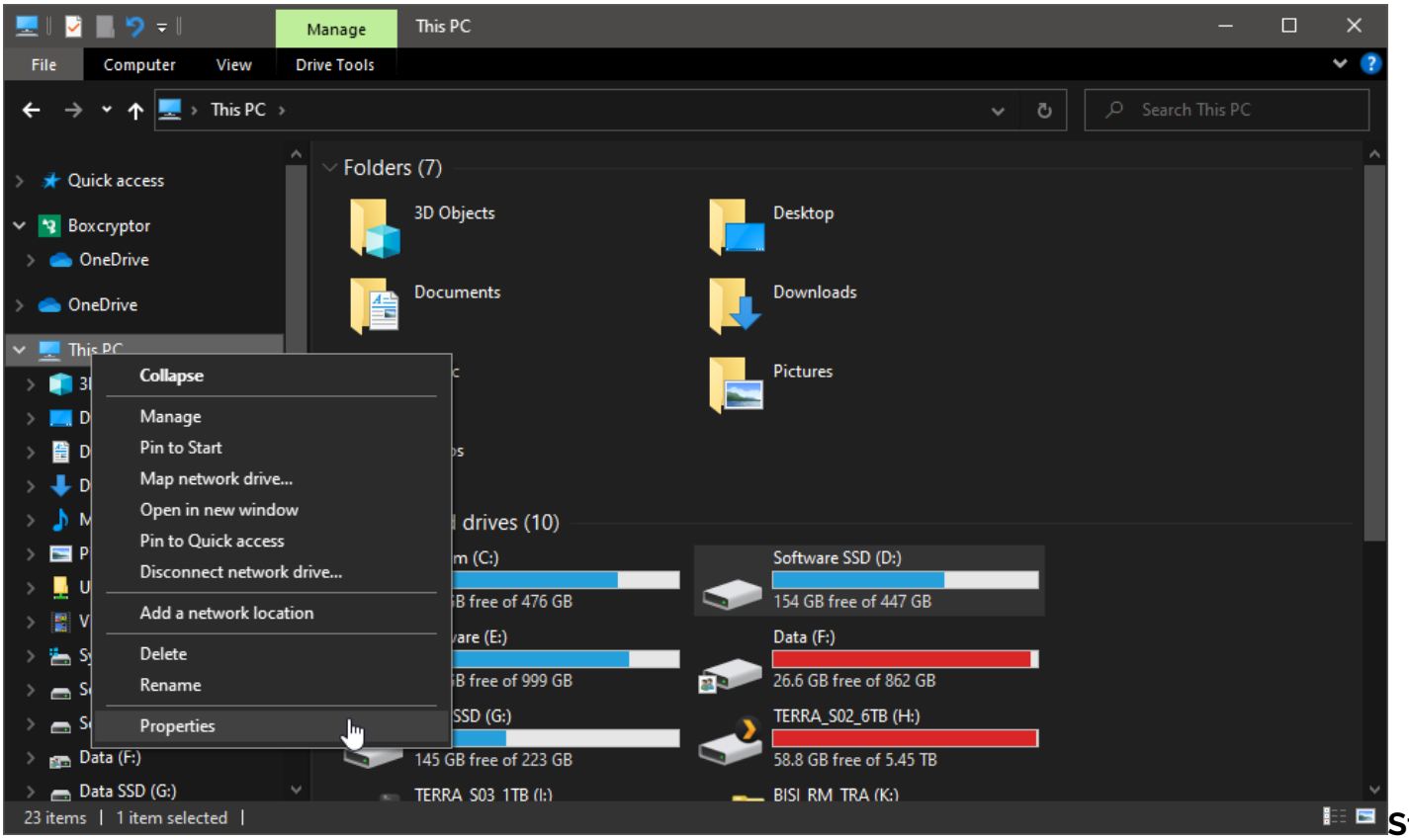
Here, we will be using the "D:\Program Files\Lua" as the location. After moving the Zip file, simply extract its content using any compression utility (Ex: Windows Explorer, 7zip, WinRar).



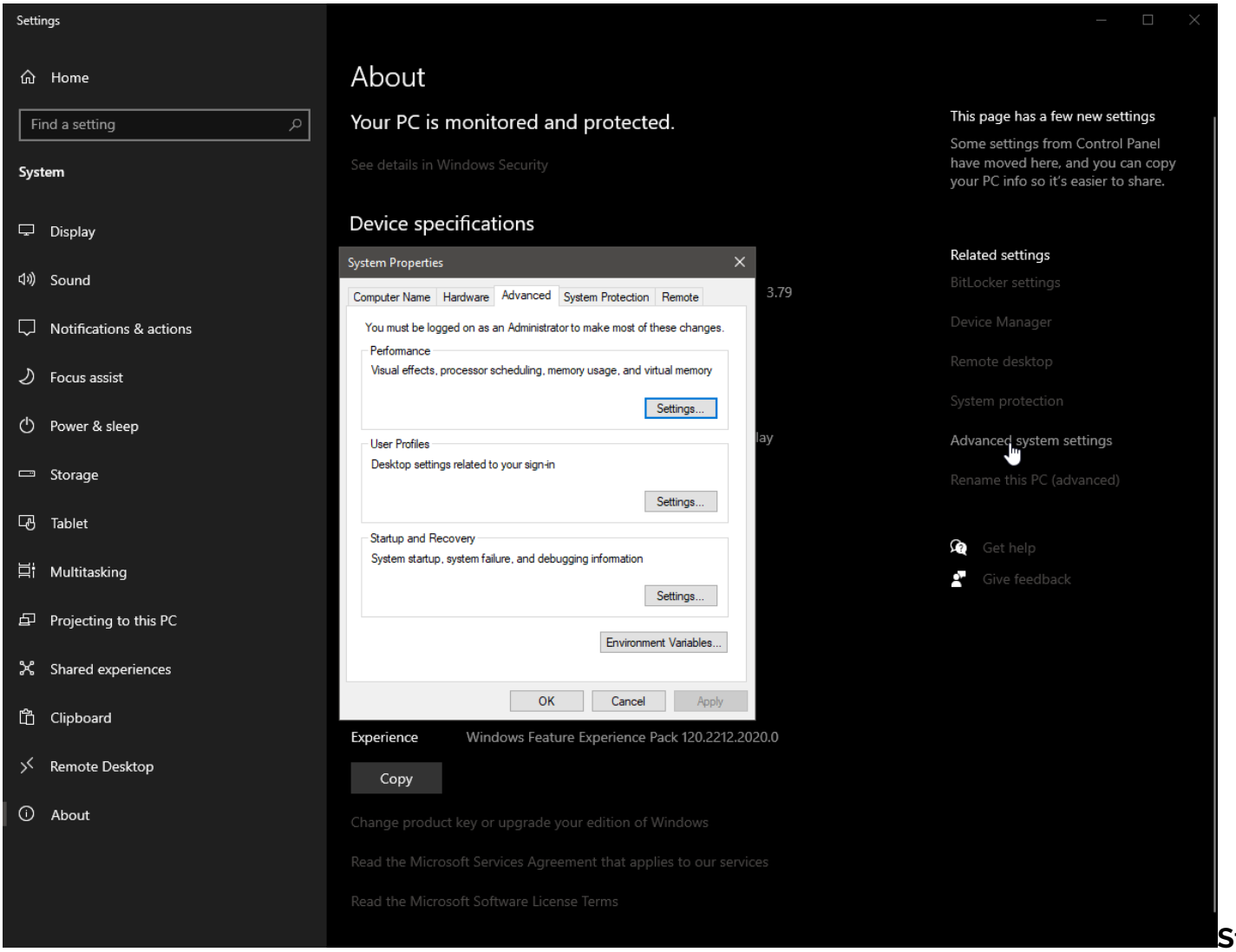
Step 4

We need to add the location of Lua binaries to the system PATH so that Windows can call Lua from anywhere in the system.

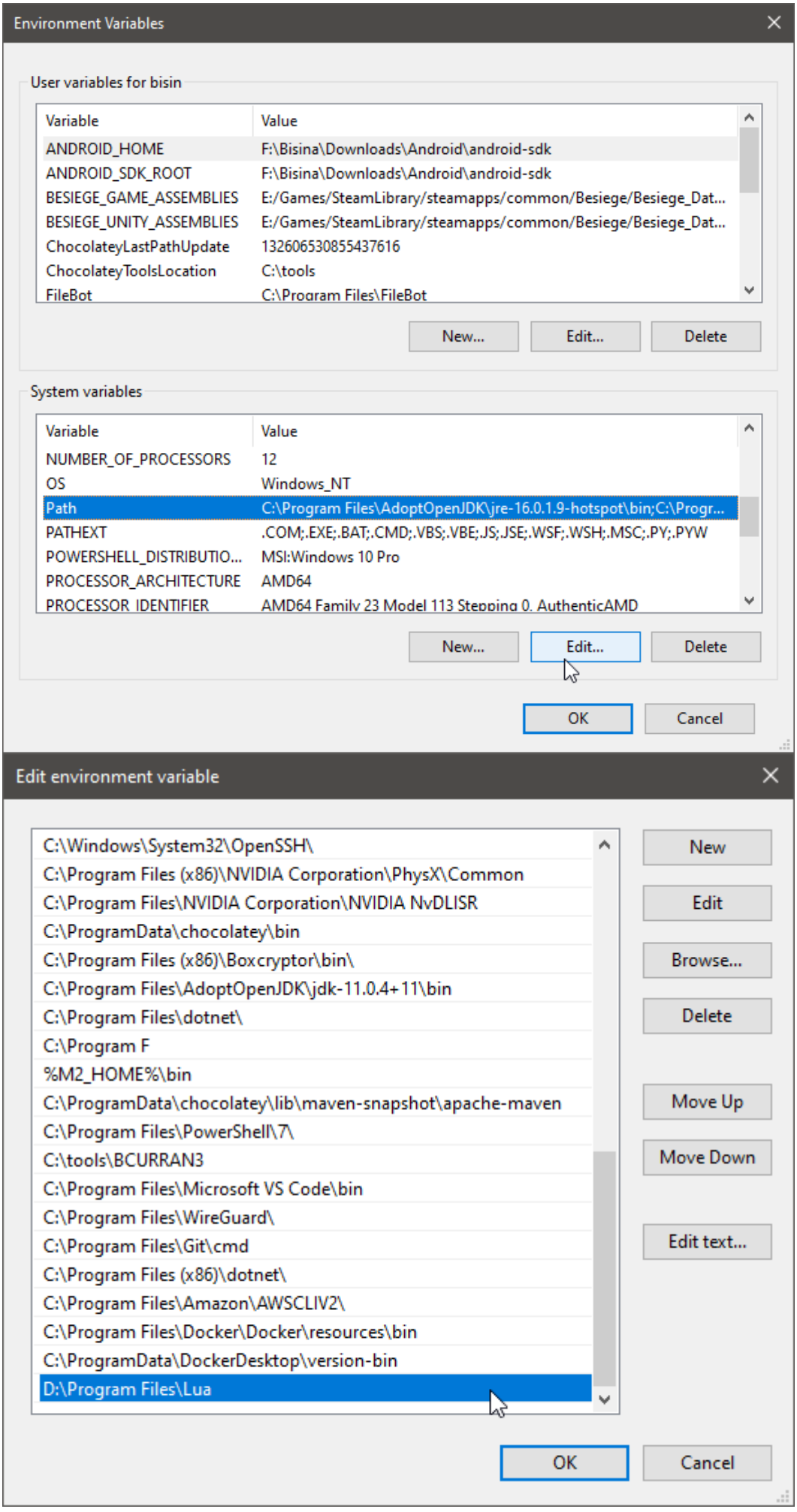
Step 4.1. Navigate to Environment Variables. (Open Windows Explorer, right-click on This PC, and select properties.)



Step 4.2. Click on “Advanced System Settings” in the screen that appears and then click on “Environment Variables”.

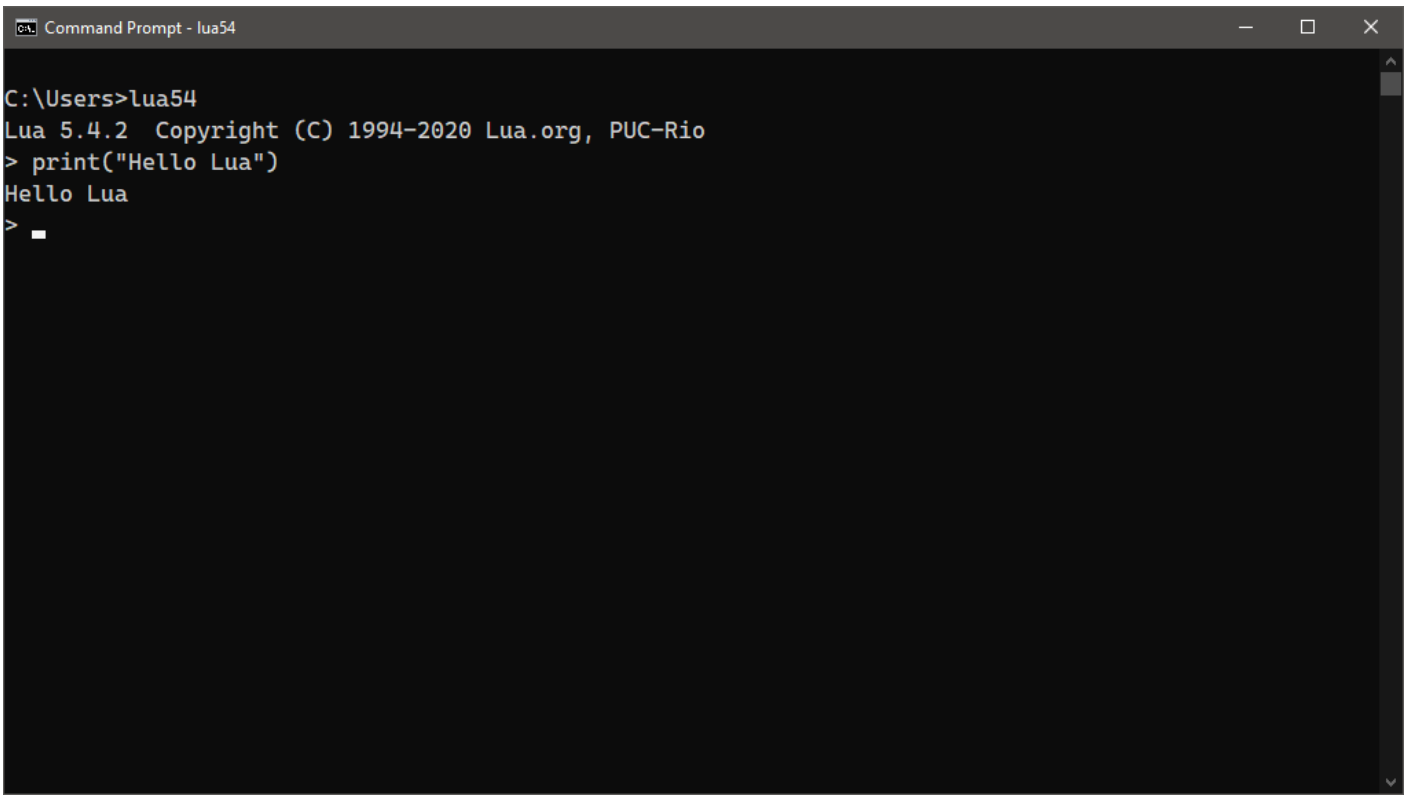


Step 4.3. In the system variables section, add the location of the Lua executables as a new entry for the Path variable.


Environment Variables
User variables for bisin
Variable
Value
ANDROID_HOME
F:\Bisina\Downloads\Android\android-sdk
ANDROID_SDK_ROOT
F:\Bisina\Downloads\Android\android-sdk
BESIEGE_GAME_ASSEMBLIES
E:/Games/SteamLibrary/steamapps/common/Besiege/Besiege_Dat...
BESIEGE_UNITY_ASSEMBLIES
E:/Games/SteamLibrary/steamapps/common/Besiege/Besiege_Dat...
ChocolateyLastPathUpdate
132606530855437616
ChocolateyToolsLocation
C:\tools
FileBot
C:\Program Files\FileBot
New...
Edit...
Delete
System variables
Variable
Value
NUMBER_OF_PROCESSORS
12
OS
Windows_NT
Path
C:\Program Files\AdoptOpenJDK\jre-16.0.1.9-hotspot\bin;C:\Progr...
PATHEXT
.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC;.PY;.PYW
POWERSHELL_DISTRIBUTIO...
MSI:Windows 10 Pro
PROCESSOR_ARCHITECTURE
AMD64
PROCESSOR_IDENTIFIER
AMD64 Family 23 Model 113 Stepping 0, AuthenticAMD
New...
Edit...
Delete
OK
Cancel
Edit environment variable
C:\Windows\System32\OpenSSH\
C:\Program Files (x86)\NVIDIA Corporation\PhysX\Common
C:\Program Files\NVIDIA Corporation\NVIDIA NvDLISR
C:\ProgramData\chocolatey\bin
C:\Program Files (x86)\Boxcryptor\bin\
C:\Program Files\AdoptOpenJDK\jdk-11.0.4+11\bin
C:\Program Files\dotnet\
C:\Program F
%M2_HOME%\bin
C:\ProgramData\chocolatey\lib\maven-snapshot\apache-maven
C:\Program Files\PowerShell\7\
C:\tools\BCURRAN3
C:\Program Files\Microsoft VS Code\bin
C:\Program Files\WireGuard\
C:\Program Files\Git\cmd
C:\Program Files (x86)\dotnet\
C:\Program Files\Amazon\AWSCLIV2\
C:\Program Files\Docker\Docker\resources\bin
C:\ProgramData\DockerDesktop\version-bin
D:\Program Files\Lua
New
Edit
Browse...
Delete
Move Up
Move Down
Edit text...
OK
Cancel

Step 5

Check if the system identifies Lua by opening up a command prompt or a PowerShell window and typing the Lua command (Lua with the version - lua54).



Setting up a Lua development environment

Now that we have installed Lua in the system, we need a development environment to go ahead with coding. For that, we can choose between:

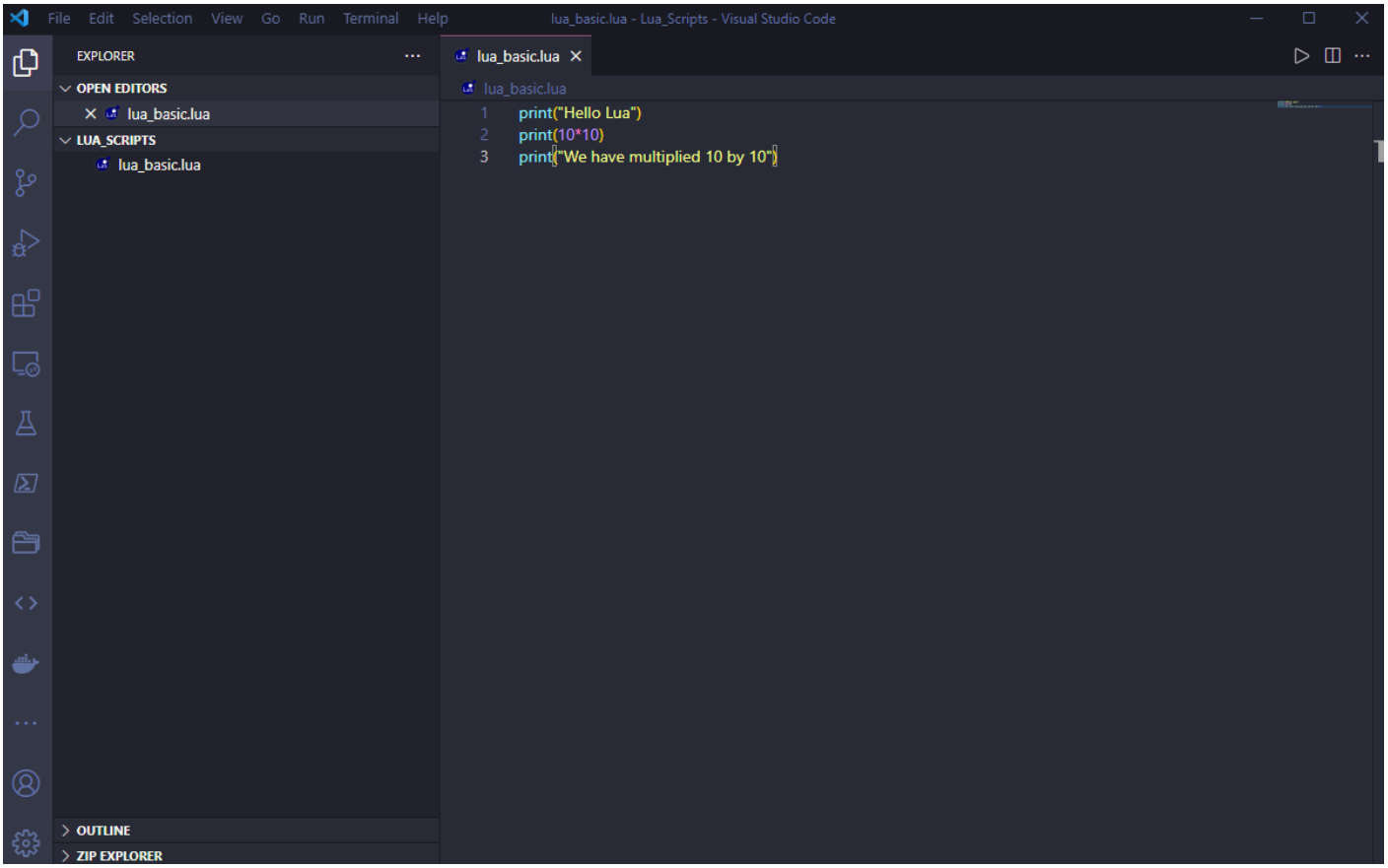
- A dedicated Lua IDE like ZeroBrane Studio
- A general IDE like VSCode

We will be using VSCode for this instance.

Step 1

Let's create a file called "lua_basic.lua" in VSCode and save that file in the desired location. Then we will type some print statements there like the following.

```
print("Hello Lua")
print(10*10)
print("We have multiplied 10 by 10")
```

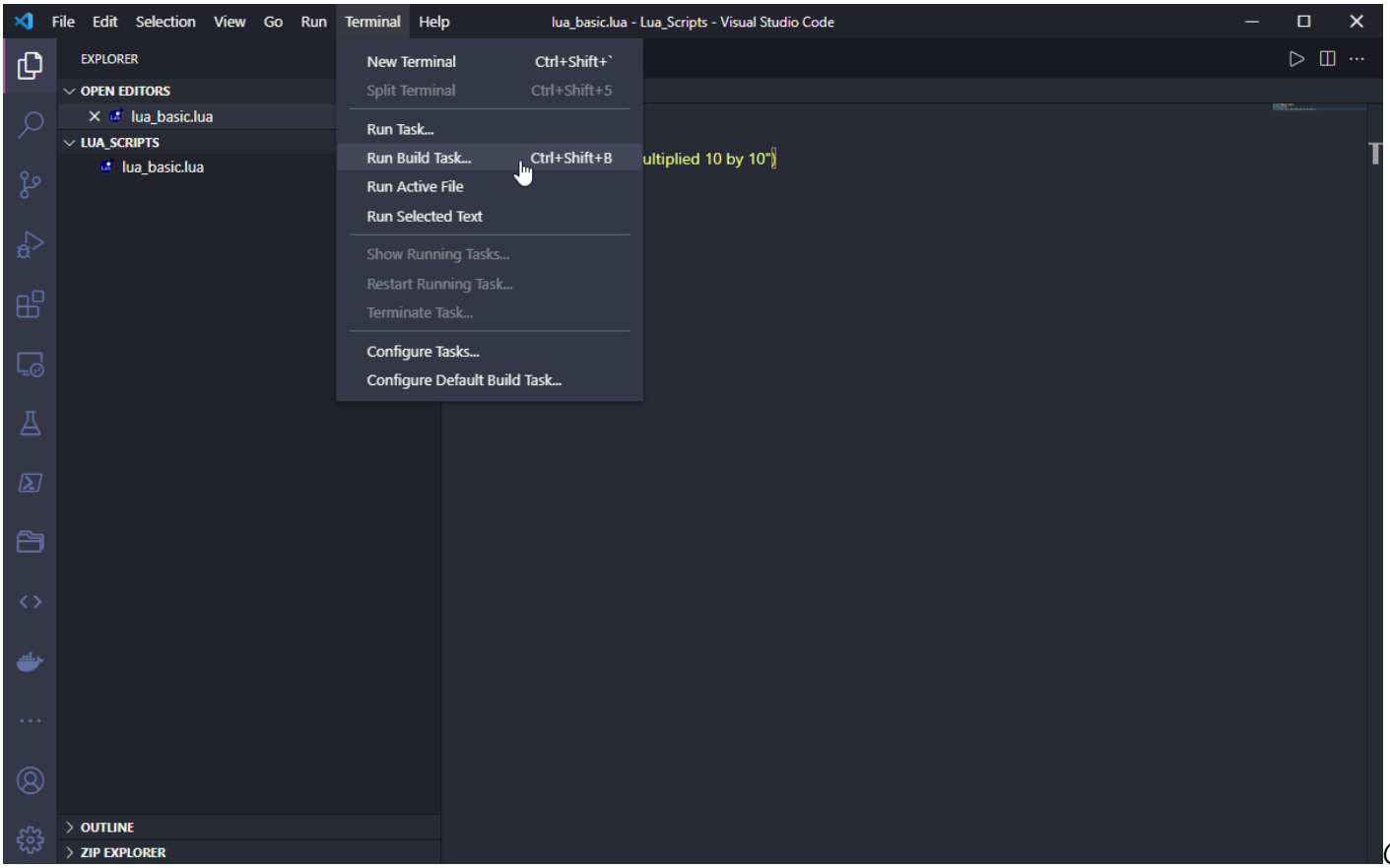


Step 2

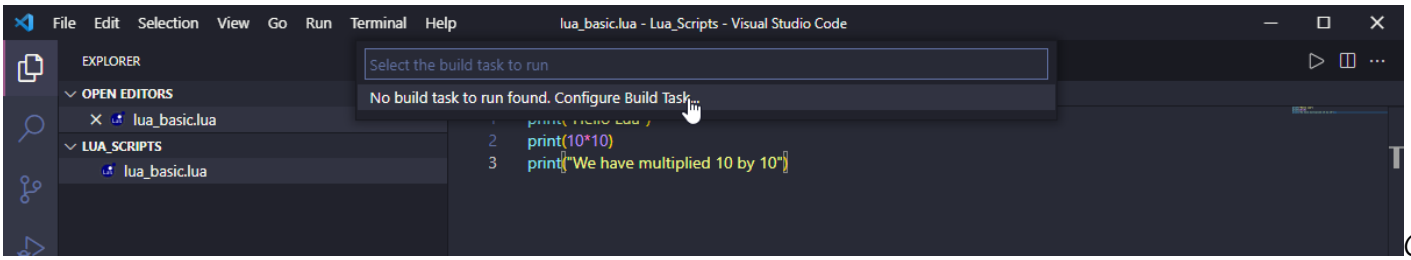
Our Lua program needs to be compiled before running, so we need to create a Build Task. To do that, click on Terminal Menu, then Run Built Task and select the Configure Build Task Option.

(There will be different build task templates depending on the VSCode configuration. Select "Create tasks.json from template" and finally the "Others" option to define a custom build task.)

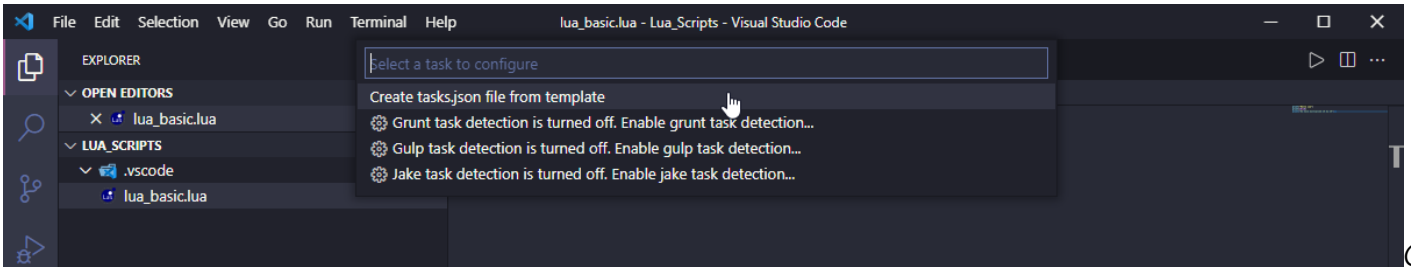
Run Build Task



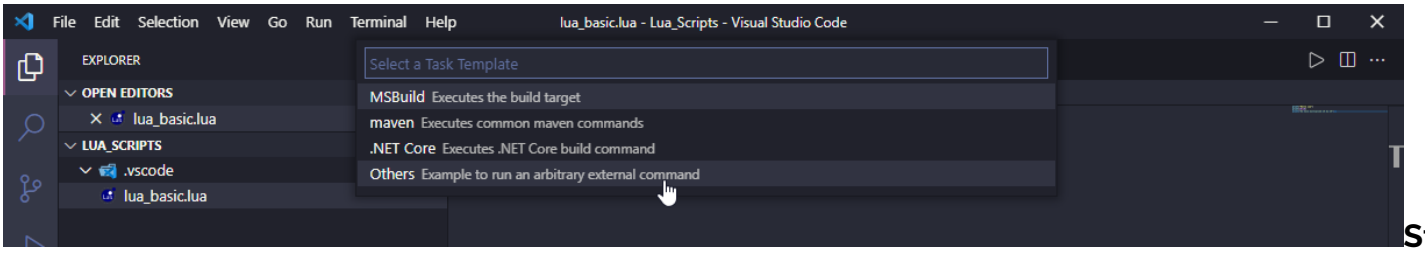
Configure Build Task



Create tasks.json from template



Other

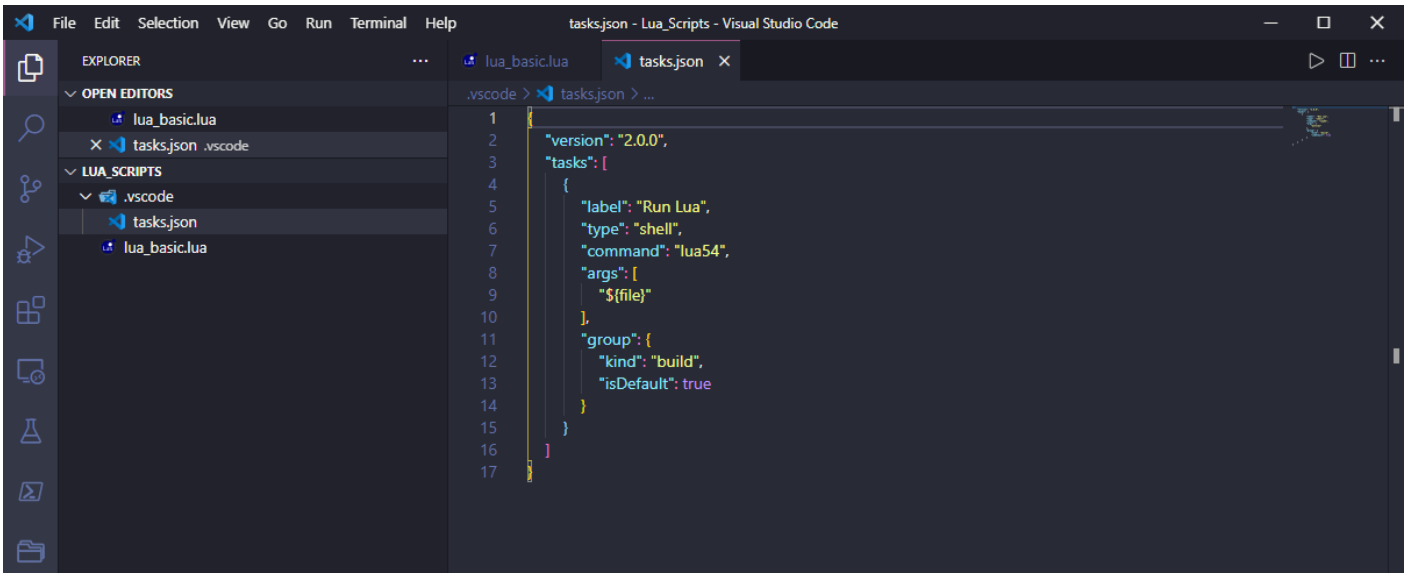


Step 2.1. We will add the following code block to configure the task. In that code block, we have defined a task called “Run Lua” that will run on the shell with the command “lua54”. That command will take the current file as the argument and carry out a build operation.

```
{
```

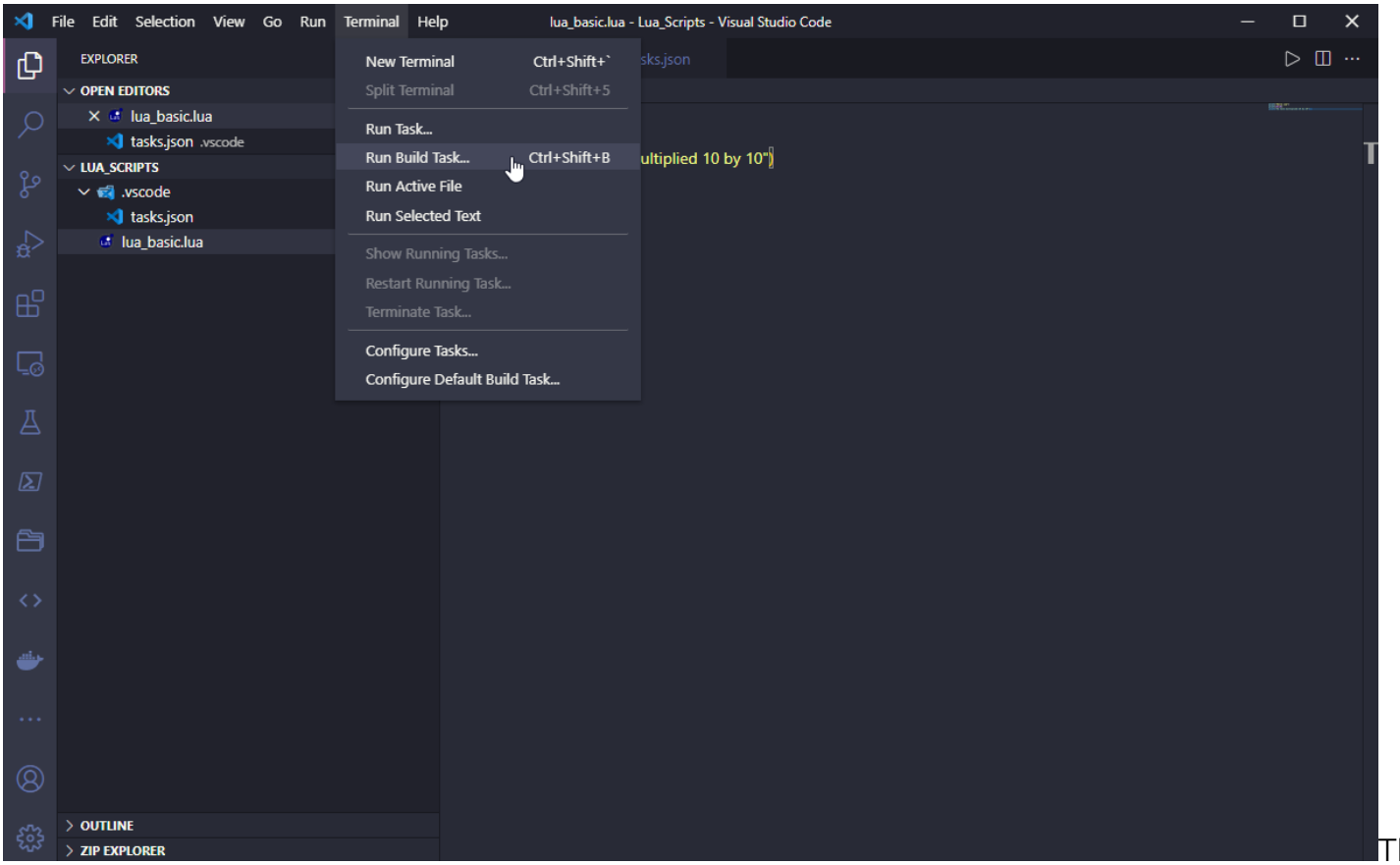
```
    "version": "2.0.0",
    "tasks": ,
            "group": {
                "kind": "build",
                "isDefault": true
            }
        }
    ]
}
```

task.json

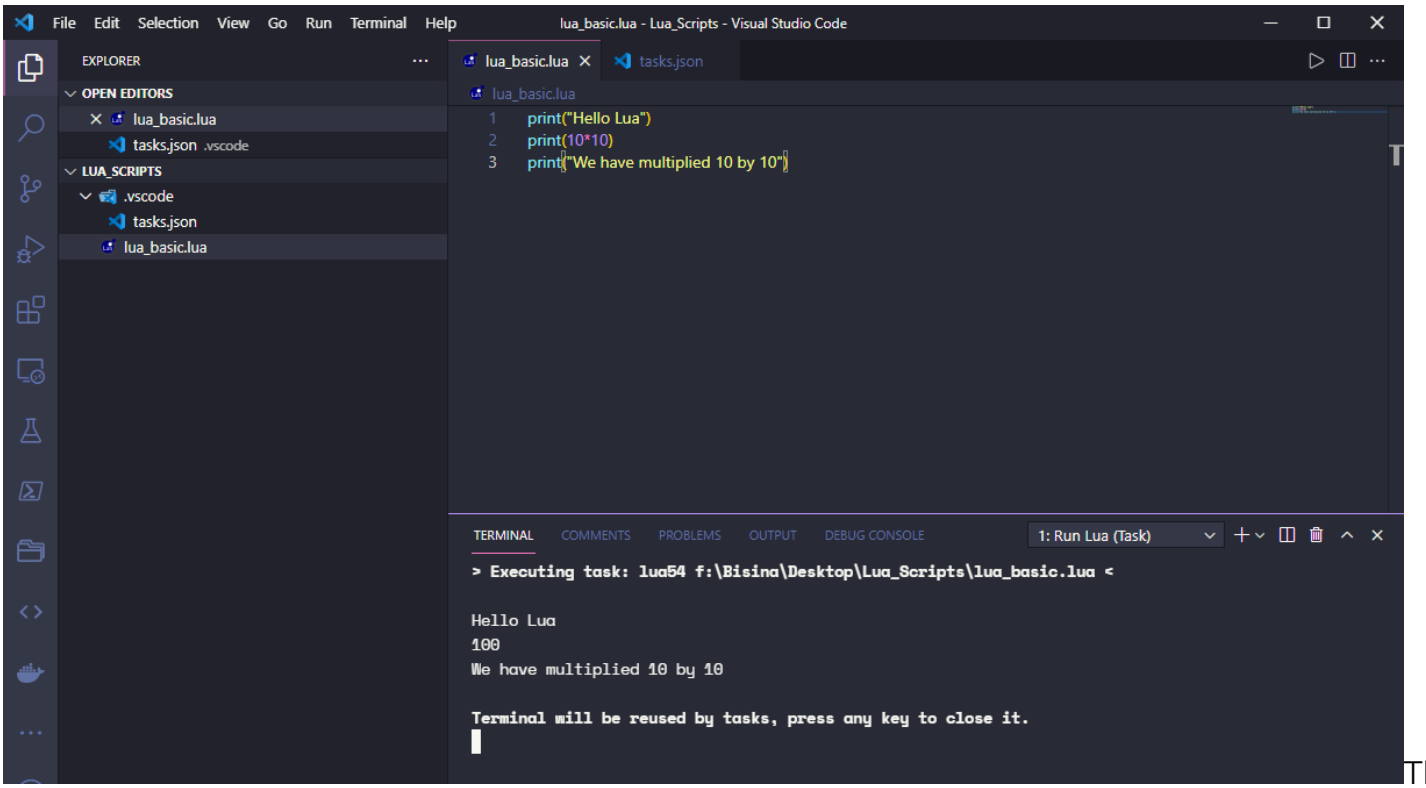


Step 3

Open up the "lua_basic.lua" file again. Then go to the "Terminal" again and click on "Run Build Task" or use the shortcut Ctrl+Shift+B.



This will compile the file and provide us with the output.



That's it! Now we have a working Lua development environment that can be used to create Lua scripts. We can use the official Lua reference manual to explore the Lua language further.

Lua is powerful

Lua is a powerful scripting language that has limitless potential to add functionality to any program

on a multitude of platforms to suit any use case.

Related reading

- BMC DevOps Blog
- Top 5 Best Practices for Software Development
- Java vs Go: What's The Difference?
- Python Development Tools: Your Python Starter Kit
- DevOps Engineer Roles & Responsibilities
- API/Developer Portals: How To Create Great API Portals