[Sollte ich Lua, Python oder Go lernen?](https://de.quora.com/Sollte-ich-Python-Lua-oder-Go-lernen)



[Hanno Behrens](https://de.quora.com/profile/Hanno-Behrens-2), Arbeitet bei Einzelunternehmen (1997-jetzt)

[Am 22. Januar beantwortet](https://de.quora.com/Sollte-ich-Python-Lua-oder-Go-lernen/answer/Hanno-Behrens-2)

**LUA**

Ich arbeite mit Lua, C und ASM als meine Hauptsprachen derzeit. Etwas OpenCL von Zeit zu Zeit, wenn es um Number-Crunching geht.

Ich hab' mal kommerziell mit Python entwickelt und kann nur jedem raten, davon die Finger zu lassen. Das Ding ist langsam wie eine am Boden festgenagelte Nacktschnecke, zäh wie gekühlter Honig und verast Speicher fast so schlimm wie Java.

Go ist lächerlich, ich hab damit kompiliert, das werden gigantische Codeblöcke und ewig langsam ist das auch noch. Kein Interesse.

Nachdem ich mit C++, Java und Python durch bin, hat es eine zeitlang gedauert, bis ich einen Ersatz gefunden habe und dieser Ersatz ist Lua(JIT). Lua geht einen völlig anderen Weg und ich habe damit meinen Abschied genommen von der Suche nach dem heiligen Gral der Programmiersprachen, ich nehme inzwischen Sprachen, die zu den Problemen passen, die ich habe, die diese Probleme effizient und damit schnell lösen und bei denen ich auch auf nichts verzichten muss.

Die editierbare Shell von Python ist nett und LuaJIT hatte sie nicht, daher hab' ich das mal kurz eingebaut, auf meinem Github kann man sich die aktuelle Version herunter laden.

[silizium/luajit-2.0](https://github.com/silizium/luajit-2.0) Ich empfehle den Branch 2.1 zu wählen, bevor man das kompiliert und selbst benutzt. Meine Version ist in vielen Dingen zwischen 15–20% schneller als das Original, aber viel mehr habe ich nicht tun können.

[LuaJIT](http://luajit.org/) kommt aus Deutschland und wurde von Michael "Mike" Pall programmiert. Die guten Lösungen auf [Rosettacode](http://rosettacode.org/) sind alle von ihm, es gab auch ein paar, die mit Lua 5.3 gearbeitet haben, aber ich lehne 5.3 als einen Schritt in die falsche Richtung ab. Der Rest der guten Lösungen ist von mir.

Nichts gegen 5.3, aber sie haben die Geschwindigkeit von 5.2/LuaJIT geopfert für ein paar unnütze kleine Features und das ist es nicht wert.

Wir entwickeln LuaJIT weiter. Was ich nicht mache, das wird unter dem Projekt [MoonJIT](https://github.com/moonjit/moonjit) entwickelt. Ich habe mich noch nicht beim Projekt gemeldet, aber werde das wohl bald unterstützen, sobald ich etwas Luft habe dafür.

Sie brauchen dringend gute Assembler-Kenntnisse und ich bin wohl einer der wenigen, die es noch gibt in Deutschland, vermutlich der einzige auf Quora, der aktiv programmiert darin.

Es gibt noch ein paar Dinge, die ich klären muss, bevor ich dem Projekt beitrete, aber das ist ein anderes Thema. Ich muss vorher ein paar Dinge ausprobieren, um LuaJIT noch schneller zu machen. Ich will hier nicht vorweg greifen und falsche Hoffnungen machen. LuaJIT ist teuflisch schnell, wenn ich C nicht auf Teufel komm raus optimiere, werde ich häufig genug nass gemacht von Lua.

Tatsächlich sind die meisten Lösungen in LuaJIT nur wenige Prozent langsamer als die selben Lösungen in C, was für eine dynamische Script-Hochsprache wie Lua allerhand ist.

Anders als Python besitzt Lua einen einfachen Syntax, den man ohne Probleme auch Anfängern zumuten kann. Es orientiert sich dabei an den Vorschlägen, die schon in de 60er Jahren mit BASIC verfolgt wurden.

Eine For-Schleife etwa ist mit

1. for i=1,10 do
2. print(i)
3. end

Klar und deutlich und wird die Zahlen 1…10 ausgeben. Arrays oder Tables wie sie in Lua heißen werden sinnigerweise von 1 an numeriert, nicht wie bei Speicher-orientierten Sprachen von 0.

In Python käme ich bei der For-Schleife bei Anfängern schon in Erklärungsnöte

1. for i in range(1,10):
2. print(i)

Zählt un-intuitiver Weise von 1 bis 9 und wenn ich einer Klasse von 14jährigen Programmieranfängern schon bei Einführung einer For-Schleife mit dem Oberkopf eines Generators kommen muss und was ein Generator ist und wie er arbeitet, dann kriege ich ich mindestens 29 mal das 1000-Meilenstarren und eins der Mädchen fängt üblicherweise an zu heulen.

Das ist keine Theorie. Das ist erlebte Praxis, weswegen ich ein strikter Gegner vom Einsatz so eines Schwachsinns wie Python im Bildungswesen bin.

Mit Lua hingegen hatte ich großartige Erfolge, meine Klasse hat klassische Kryptografie geschrieben, eine Enigma umgesetzt und eine SZ42 Lorenz.

[silizium/crypto](https://github.com/silizium/crypto) hier der Code dazu von meinen Schülern.

Das wäre unmöglich gewesen mit Python. Ich habe in Lua nur 60% der Keywords zu unterrichten, also 22 statt äh 36? ich glaube ja. Und die Konzepte sind alle einfacher und flacher und man drückt ihnen das Gesicht nicht von der ersten Stunde an in den stinkenden Haufen von objektorientierter Programmierung, der sich als völliger Irrweg herausgestellt hat und in der Praxis längst von datenorientierter Programmierung abgelöst wurde.

Also zurück zu den Anfängen der Prozeduralen Programmierung im berühmten 1968er in Garmisch-Partenkirchen.

[Strukturierte Programmierung – Wikipedia](https://de.wikipedia.org/wiki/Strukturierte_Programmierung)

Aus den Feststellungen hat die OOP nicht heraus geführt, im Gegenteil, unsere Probleme sind schlimmer denn je.

In Lua kann man OOP programmieren, aber die Sprache ist frei von Paradigmen, frei von Ideologie und hat statt dessen eine freie "Metatable", die es einem erlaubt die eigenen Anforderungen an die Software zu definieren, also notfalls Klassen und Vererbung und meinetwegen Multi Vererbung, wenn man es nicht lassen kann.

Aber im Gegensatz zu Python bietet Lua dabei einen richtigen Schutz privater Methoden oder Attribute, statt eines stumpfen Generators implementiert Lua die sehr viel mächtigere echte Coroutine, statt des inzestuös-zurückgebliebenen Kindes des Lambda-Kalküls von Python unterstützt Lua echte funktionale Programmierung. Zudem blockiert nicht wie bei Python ein globaler Interpreterlock jedes Multithreading im Ansatz, mit POSIX läßt sich problemlos parallel programmieren und die Systemintegration ist derartig perfekt, dass es die einzige Hochsprache ist, mit der sich ein Linux-Kernel bootstrappen ließe.

Lua ist dabei mindestens 36 mal so schnell wie Python, was häufiger zu Faktor 100 führt oder in einem geradezu satirischen Fall Faktor 42000 zu Python bedeutet.

Dabei unterstützt Lua natürlich AI Programmierung, mit Torch sogar die selben Binär-Libraries die auch Python nutzt, nur halt sehr viel schneller.

LuaJIT 42000 mal schneller als Python: [Can you show me some Lua code?](https://www.quora.com/Can-you-show-me-some-Lua-code/answer/Hanno-Behrens-2)

Was bei Python kaputt ist hier: [https://www.quora.com/How-do-you...](https://www.quora.com/How-do-you-count-characters-in-Python/answer/Hanno-Behrens-2/comment/118543229)

Was bei OOP kaputt ist hier: [https://www.quora.com/Which-prog...](https://www.quora.com/Which-programming-language-is-fully-object-oriented/answer/Aaron-Christianson-2/comment/124606996)

Von Go die Finger lassen, das ist einfach lächerlich. Das ist noch in den Babyschuhen und macht sich noch in die Windeln. Schon der Compile-Vorgang ist grauenhaft. Finger weg!

Wer mit Lua anfangen will, kann das sehr leicht probieren mit der freien [IDE Zerobrane](https://studio.zerobrane.com/download?not-this-time), er kann [LuaJIT](http://luajit.org) herunter laden, es gibt über "luarocks" eine feine Bibliothek für lauter Probleme, die zwar nicht ganz so umfangreich ist wie die von Python, aber seien wir ehrlich der Pythonkram funktioniert häufig nur auf Python 2 oder ist das Produkt eines Programmieranfängers und ist dann auch am Ende so nützlich wie ein Kropf.

Die meisten dieser Bibliotheken sind erfahrungsgemäß untereinander inkompatibel und der Aufwand ist so extrem, dass man ein "venv" braucht, um künstliche Laufzeit-Environments anzulegen für Python. Es ist grauenhaft.

Und als Programm verschiffen läßt sich Python auch nicht vernünftig, der Aufwand ist extrem und das Ergebnis nicht einmal mittelmäßig.

Lua hingegen versucht nicht als heiliger Gral zu gelten die meisten der Bibliotheken für Lua sind direkt in C geschrieben, wie das geht wird bereits im [Referenz-Manual](https://www.lua.org/manual/5.2/contents.html#contents) erklärt, daher sind diese professionell und hoch effizient und liegen qualitativ in der Regel auf einem völlig anderen Niveau.

Wer Spiele schreiben will, kann dies mit Lua/[Löve2D](https://love2d.org/wiki/Main_Page) tun. Aber es gibt auch weitere Game-Engines wie [Urho3D](https://urho3d.github.io/) etwa und viele andere. Als günstige Veröffentlichungs-Plattform bietet sich <https://itch.io> an, die von Lua Programmierern gegründet wurde, aber inzwischen als einer der Haupt-Anlaufpunkte für Indie Entwicklungen gilt.

Eine Menge Spiele nutzen Lua als Script-Sprache, dazu gehört Minecraft, [Minetest](https://minetest.net), RobLox, WoW, CryEngine und viele andere.

Die Sache ist die, dass Lua sich ohne Probleme mit nur zwei Zeilen in jedes bestehende Projekt einbinden läßt und so diese Apps intelligent macht, sozusagen "App 2.0". Dafür gibt es jetzt immer mehr und mehr Beispiele, NMap, Wireshark, Vim und Neo-Vim, LuaLaTeX und und und, die Liste wird beinahe täglich länger.

Und das, was man meisten für Lua spricht ist, die Geschwindigkeit und die winzige Größe mit nur 400 bis 800 KILObytes. Und das läßt den Einsatz auf Kleinstcomputern wie dem Raspberry Pi zu (in allen Versionen), das erlaubt eben auch hoch effiziente Webseiten zu programmieren in Lua als Serversprache wie bei [Mako/Barracuda](https://makoserver.net/), wo es auch ein nettes, kleines Tutorial gibt.

Meine eigene Beispielseite: <http://ratsnest.spdns.de:4180/> Die Seite reagiert zehnmal schneller mit dynamischen Seiten als Apache es mit statischen schafft. Und dabei läuft sie nur auf einem alten RPi.

Lernen kann man natürlich auch auf [Tutorialspoint](https://www.tutorialspoint.com/lua/index.htm) und anderen entsprechenden Seiten, die überall wie Pilze aus dem Boden schießen, sowie über mehrere Bücher und meinem eigenen über Kryptografie in Lua, das ich aber noch nicht veröffentlicht habe, aber auf Anfrage kann ich PDF Kopien versenden.

Danach kann man komplett lernen und es richtet sich eben an Anfänger.

Der Rest ist jetzt aktiv werden. Und damit natürlich auch lokale Entwickler unterstützen, LuaJIT ist ein deutsches Produkt. Schiebt mal dem Mike Pall ein paar Euro rüber. Er ist ein Genie.

Ich hoffe, ich konnte helfen.