Lua for Programmers Part 4: Tips and Tricks

Posted on Sep 09, 2012 in Tutorials

If you haven't read the previous parts (one, two and three), then I'd highly recommend doing so. For reference, here's a list of the parts in this series:

- Part 1: Language Essentials, covers fundamental syntax and concepts such as operators, loops, and functions.
- Part 2: Data and Standard Libraries, covers Lua's built-in data types and some of the standard libraries.
- Part 3: More Advanced Concepts, deals with things like variable scope, advanced functions, and file loading.
- Part 4: Tips and Tricks, the current part; a collection of small things that you may find useful.

Command Line Arguments

Command line arguments are stored in the `arg` table. For example, if we had the script `foo.lua`:

```
print(arg[-1], arg[0])
for i, v in ipairs(arg) do print(v) end
```

And we ran it via `lua foo.lua arg1 arg2 arg3`, the output would be:

```
lua foo.lua
arg1
arg2
arg3
```

`arg[-1]` is the name of the interpreter/program, generally `lua`. `arg[0]` is the name of the file run. All entries after this are the command line arguments; this allows you to iterate over them as shown in the example.

`...` in Files

Since files are loaded as functions, it makes sense that you can use `...` inside of them. If we have a file called `bar.lua`:

```lua
print(...) -- prints all arguments given to this file's function
```

And then we load it like this:

```lua
loadfile("bar.lua")(1, 2, 3, 4)
```

The output will be `1 2 3 4`.

As for the other functions, `dofile` sends no arguments, and `require` sends a single argument with the path it was given:

```lua
require("bar") -- "bar" is the output
require("folder.subfolder.bar") -- "folder.subfolder.bar" is the output
```

If you use `...` in the entry-point file (this would be `foo.lua` if you executed `lua foo.lua`, for example) you'll get a list of command line arguments. So if we were to run `bar.lua` via `lua bar.lua arg1 arg2 arg3`, the output would be `arg1 arg2 arg3`.

_G

`_G` is a global table which holds all global variables. Here's an example:

```lua
a = 3
print(_G.a) -- 3
_G.b = 4
print(b) -- 4
print(_G._G == _G) -- true
```

Conclusion

Well, that's it for now. If you've got any suggestions for things that could be added, I'd love hear them.

Thanks for reading the series, I hope it's been of use to you.

programming lua

« Lua for Programmers Part 3: More Advanced Concepts | Custom Cursors in Love2D »

Latest Posts

Manually Installing phpMyAdmin (Ubuntu 18.04)
Jan 17, 2019

Installing an Up-To-Date LAMP Stack (Ubuntu 18.04)
Jan 12, 2019

Projects Over the Last Few Years
Jul 17, 2018

Website Overhaul
Jul 10, 2018

Ludum Dare 27: Triad
Aug 29, 2013

Categories

Tutorials (27)

Projects (21)

General (4)

Posts by Year

2019 (2)

2018 (2)

2013 (1)

2012 (12)

2011 (34)

2010 (1)