The Lua Programming Language

Nick Shannon

What is Lua?

- Created in 1993 by a group at the University of Rio de Janeiro, Brazil
- Lightweight, high-level, multi-paradigm
- Designed with speed and portability in mind

What is Lua?

- Interpreted, dynamically-typed scripting language
- Fast and small:
  - The fastest (or one of the fastest) scripting languages around
  - Interpreter and standard libraries are 281K
- Ideal for embedding into apps

The Lua Interpreter

- Lua has a stand-alone interpreter, `lua`
- We can use Lua as a script interpreter in Unix systems by adding `#!/bin/lua` to the first line of a file

Executing the Interpreter

```
[nick@home ~]$ lua
```

The Lua Interpreter

- Lua has a stand-alone interpreter, `lua`
- We can use Lua as a script interpreter in Unix systems by adding `#!/bin/lua` to the first line of a file

Executing the Interpreter

```
[nick@home ~]$ lua
> print("hello")
```

The Lua Interpreter

- Lua has a stand-alone interpreter, `lua`
- We can use Lua as a script interpreter in Unix systems by adding `#!/bin/lua` to the first line of a file

Executing the Interpreter

```
[nick@home ~]$ lua
> print("hello")
hello
```

The Lua Interpreter

- Lua has a stand-alone interpreter, `lua`
- We can use Lua as a script interpreter in Unix systems by adding `#!/bin/lua` to the first line of a file

Executing the Interpreter

```
[nick@home ~]$ lua
> print("hello")
hello
> x = 10
```

The Lua Interpreter

- Lua has a stand-alone interpreter, `lua`
- We can use Lua as a script interpreter in Unix systems by adding `#!/bin/lua` to the first line of a file

Executing the Interpreter

```
[nick@home ~]$ lua
> print("hello")
hello
> x = 10
> print(x)
```

The Lua Interpreter

- Lua has a stand-alone interpreter, `lua`
- We can use Lua as a script interpreter in Unix systems by adding `#!/bin/lua` to the first line of a file

Executing the Interpreter

```
[nick@home ~]$ lua
> print("hello")
hello
> x = 10
> print(x)
10
```

The Lua Interpreter

- Lua has a stand-alone interpreter, `lua`
- We can use Lua as a script interpreter in Unix systems by adding `#!/bin/lua` to the first line of a file

Executing the Interpreter

```
[nick@home ~]$ lua
> print("hello")
hello
> x = 10
> print(x)
10
> print(y)
```

The Lua Interpreter

- Lua has a stand-alone interpreter, `lua`
- We can use Lua as a script interpreter in Unix systems by adding `#!/bin/lua` to the first line of a file

Executing the Interpreter

```
[nick@home ~]$ lua
> print("hello")
hello
> x = 10
> print(x)
10
> print(y)
nil
```

The Lua Interpreter

- We can do a lot just from the prompt, including loading a file and executing code directly

Advanced Example

```
[nick@home ~]$ echo 'print("hello")' > example.lua
[nick@home ~]$ lua -i -l example -e "x = 10"
hello
> print(x)
10
```

Reserved Keywords

- Lua is dynamically-typed
- There are no type definitions; each value carries its own type
- There are 8 basic types:

```
string    boolean   number  nil
function  userdata  thread  table
```

- Lua has just 21 unique keywords:

```
and       break     do        else      elseif
end       false     for       function  if
in        local     nil       not       or
repeat    return    then      true      until   while
```

Variables

- Unlike Python, Lua is not space sensitive
- The following lines are equivalent:

Initializing Variables

```
a = 1
b = 2

a = 1;
b = 2;

a = 1 ; b = 2

a = 1   b = 2 -- Please don't do this...
```

Tables

- Lua has a single data structure: **tables**

Tables

- Lua has a single data structure: **tables**
- A table is implemented as an *associative array*
  - Analogous to a map, dictionary, or symbol table

Tables

- Lua has a single data structure: **tables**
- A table is implemented as an *associative array*
  - Analogous to a map, dictionary, or symbol table
- Tables can represent arrays, maps, sets, queues, graphs, and more

Tables

- Lua has a single data structure: **tables**
- A table is implemented as an *associative array*
  - Analogous to a map, dictionary, or symbol table
- Tables can represent arrays, maps, sets, queues, graphs, and more

Tables

Initializing a table as a simple array

```
t = {"I", "love", "Lua"}
t[3] = "Lua" -- Lua uses 1-based indexing
```

Table methods

Method	Purpose
table.concat	Concatenates the strings in the tables based on the parameters given.
table.insert	Inserts a value into the table at specified position.
table.maxn	Returns the largest numeric index.
table.remove	Removes the value from the table.
table.sort	Sorts the table based on optional comparator argument.

Functions

Example Function

```
--[[
A multi-line comment.
This function splits a string into a list of words.
--]]
function Utils.split(inputstr, sep)
    if not sep then
            sep = "%s"
    end
    local t={}
    for str in string.gmatch(inputstr,
        "([^"..sep.."]+)")
    do
            table.insert(t, str)
    end
    return t
end
```

Metatables

- Metatables are a powerful feature that allow one to modify the behavior of tables
  - They come with a number of metamethods
- Useful for inheritance, enforcing types, and much more

Classes in Lua

- No built-in `class` keyword, but OOP can be emulated using functions and tables

Example "Class" in Lua

```lua
local Format = {}
function Format:new(param1)
    o = {}
    setmetatable(o, self)
    self.__index = self
    -- Put all your class variables here...
    self.param1 = param1
    return o
end
-- Class functions here...
return Format
```

Lua in the Wild

- Lua is very popular in the game development space:
  - Two notable Lua users are Activision Blizzard and Roblox Corporation
- Lua is also used for extensibility in other non-game applications:
  - Neovim - text editor
  - Redis - a key-value database
  - Nginx - a web server
  - Wireshark - network packet analyzer

Language Evaluation

- Pros:
  - Readable/writable
  - Small and portable
  - Native support for coroutines and multi-threading
  - Fast (for a scripting language)
  - Excellent ecosystem (Luarocks package manager, community support, etc.)
  - Used in a large number of projects
- Cons:
  - Very minimal feedback from the interpreter, can make debugging difficult
  - Small standard library means that many core features may be missing
  - Global scoping by default
  - Difficult to determine the shape or size of a table

Live Demo

- Demo of split()