Lua

CSCE 330 – Programming Language Structures

Presented by Steven Dindl and Trent Braley

Material Covered

History, Overview, Syntax

03. Lua's Origin
04. What is Lua?
05. Syntax & Basic Features (Code Example)

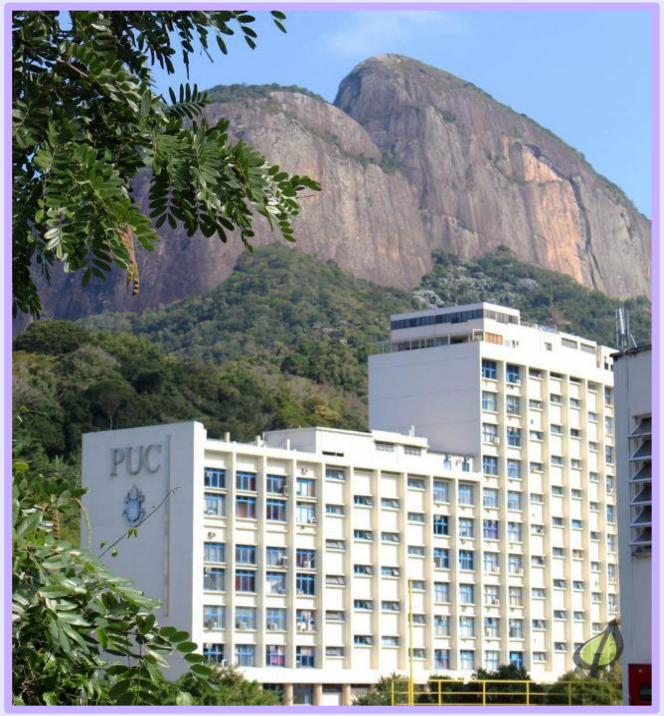
Versions, Users, Dialects

06. 1990's Releases
07. 2000+ Releases
08. Who uses Lua?
09. Lua-Based Languages (Compile to/Dialects)

Various Code Examples

10. Types
11. Tables
12. OOP with Tables/Metatables
13. C Embedding

Lua’s Origin



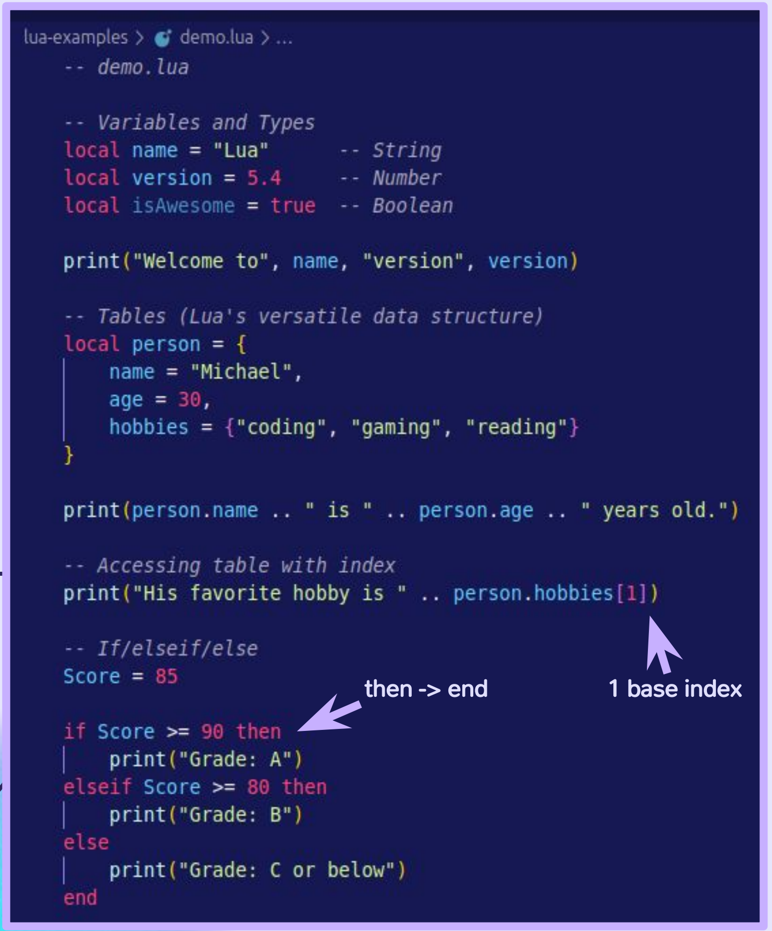
- Created in 1993 at Tecgraf (PUC-Rio, Brazil)
- Developed to simplify data entry and report generation at Petrobras (petroleum company)
- Evolved from two custom languages (DEL and Sol)
- Guided by simplicity and real user needs

What is Lua?

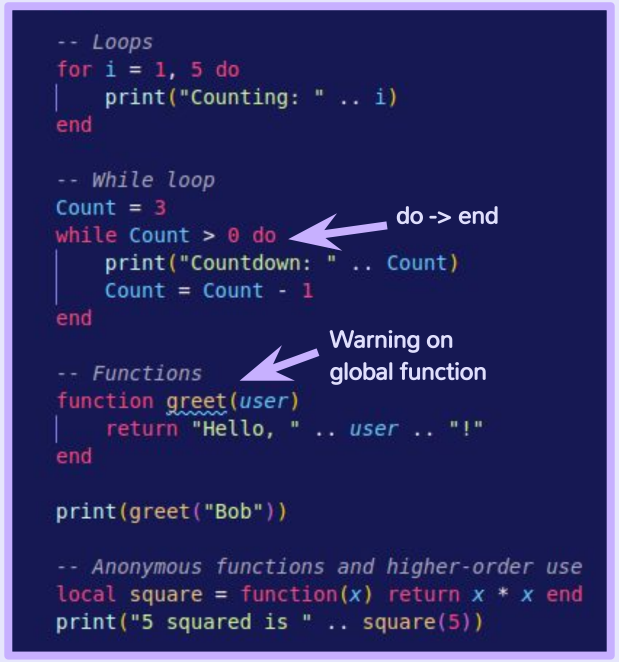
Lua...

- is a **lightweight**, **embeddable** scripting language.
- is **imperative**, **dynamically typed**, and **interpreted** (with optional compiling)
- features **minimal syntax** and is easy to learn

Code Example: Syntax & Basic Features

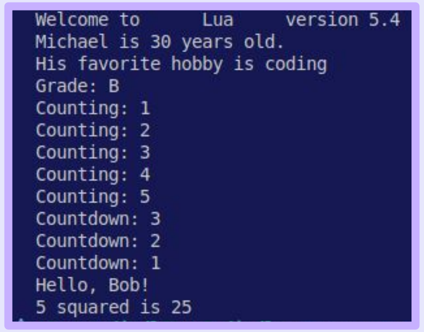


```lua
-- demo.lua

-- Variables and Types
local name = "Lua"       -- String
local version = 5.4      -- Number
local isAwesome = true   -- Boolean

print("Welcome to", name, "version", version)

-- Tables (Lua's versatile data structure)
local person = {
    name = "Michael",
    age = 30,
    hobbies = {"coding", "gaming", "reading"}
}

print(person.name .. " is " .. person.age .. " years old.")

-- Accessing table with index
print("His favorite hobby is " .. person.hobbies[1])

-- If/elseif/else
Score = 85

if Score >= 90 then
    print("Grade: A")
elseif Score >= 80 then
    print("Grade: B")
else
    print("Grade: C or below")
end
```



```lua
-- Loops
for i = 1, 5 do
    print("Counting: " .. i)
end

-- While loop
Count = 3
while Count > 0 do
    print("Countdown: " .. Count)
    Count = Count - 1
end

-- Functions
function greet(user)
    return "Hello, " .. user .. "!"
end

print(greet("Bob"))

-- Anonymous functions and higher-order use
local square = function(x) return x * x end
print("5 squared is " .. square(5))
```

To run the script:
lua demo.lua



```
Welcome to      Lua     version 5.4
Michael is 30 years old.
His favorite hobby is coding
Grade: B
Counting: 1
Counting: 2
Counting: 3
Counting: 4
Counting: 5
Countdown: 3
Countdown: 2
Countdown: 1
Hello, Bob!
5 squared is 25
```

1990's Significant Releases

Lua 1.1
1994
First public release, simple syntax, and a bytecode virtual machine

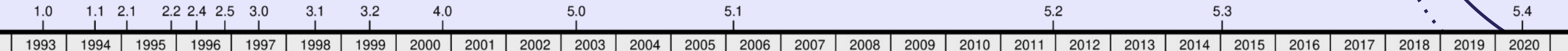
Lua 2.1
1995
OOP support and extensible semantics

Lua 2.4
1996
Luac compiler

Lua 2.5
1996
Pattern matching and vararg functions

Lua 3.0
1997
Tag (precursor to what is now metatables) and auxlib

Lua 3.1
1998
Anonymous functions, functional programming features, double precision numbers



2000+ Significant Releases

Lua 4.0
2000-2002

Multiples states, new API, for loops, full speed execution

Lua 5.0
2003-2006

Collaborative multithreading, metatables, booleans, proper tail calls, weak tables

Lua 5.1
2005-2012

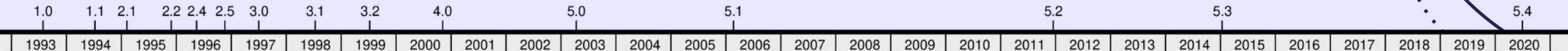
Incremental garbage collection, long strings/comments, mod op, length op, metatables for all types, luaconfig.h

Lua 5.3
2015-2020

Integers, support for both 64-bit and 32-bit platforms

Lua 5.4
2020-NOW

Generational mode for garbage collection, const and to-be-closed variables



Who uses Lua and how?

Configuration & Extension

Wireshark
Users can use lua for prototyping

NeoVim
Config, plugins

World of Warcraft
Non-gameplay UI

Core Functionalities

Adobe Lightroom
˜40% of codebase in Lua

Sims 4
Game constants, UI, tutorials, in-game behavior

FarCry
All game events and AI/game logic

Primary Language

Roblox
All game scripting on the platform

Garrys Mod
All gameplay scripting and mods

LOVE 2D
Game engine built entirely around Lua

Lua-based languages

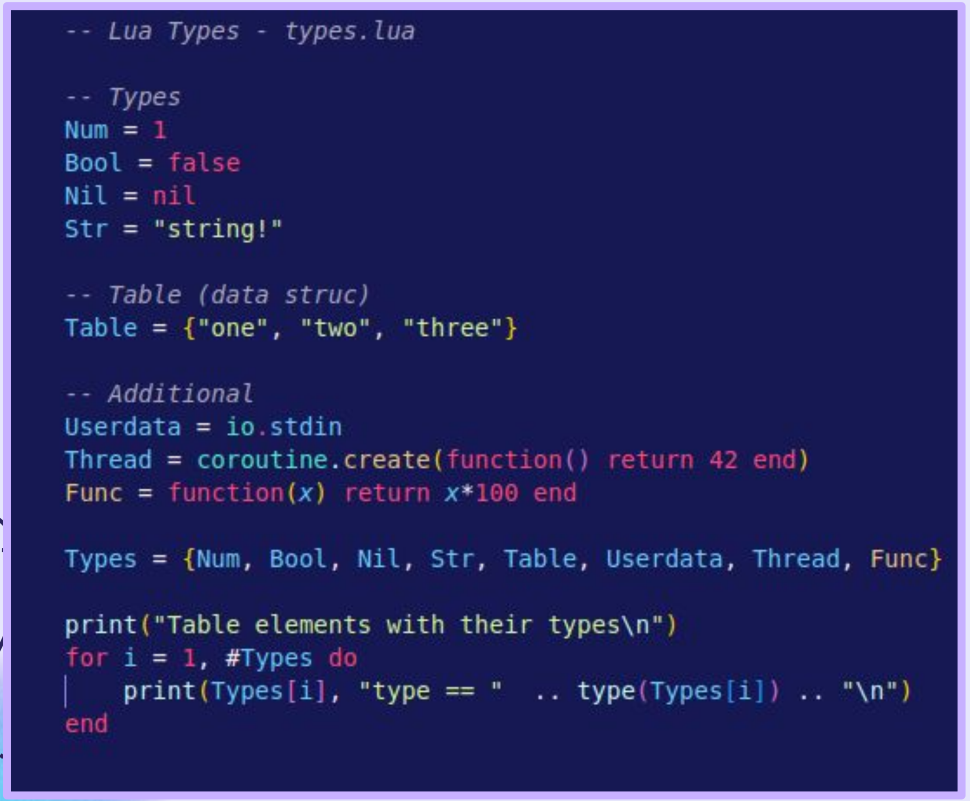
Compile to Lua

- MoonScript
- Fennel
- Team
- TypeScriptToLua

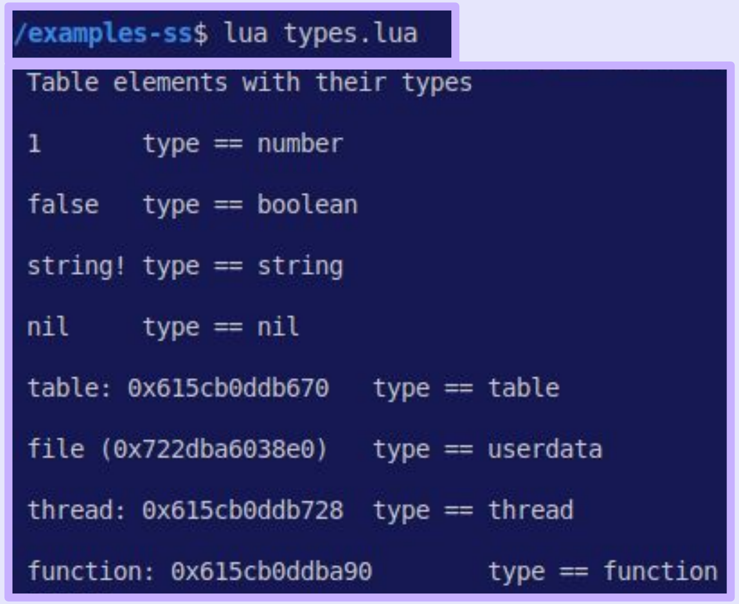
Dialects of Lua

- Luau
- Garry’s Mod Lua
- LuaJIT

Code Example: Types

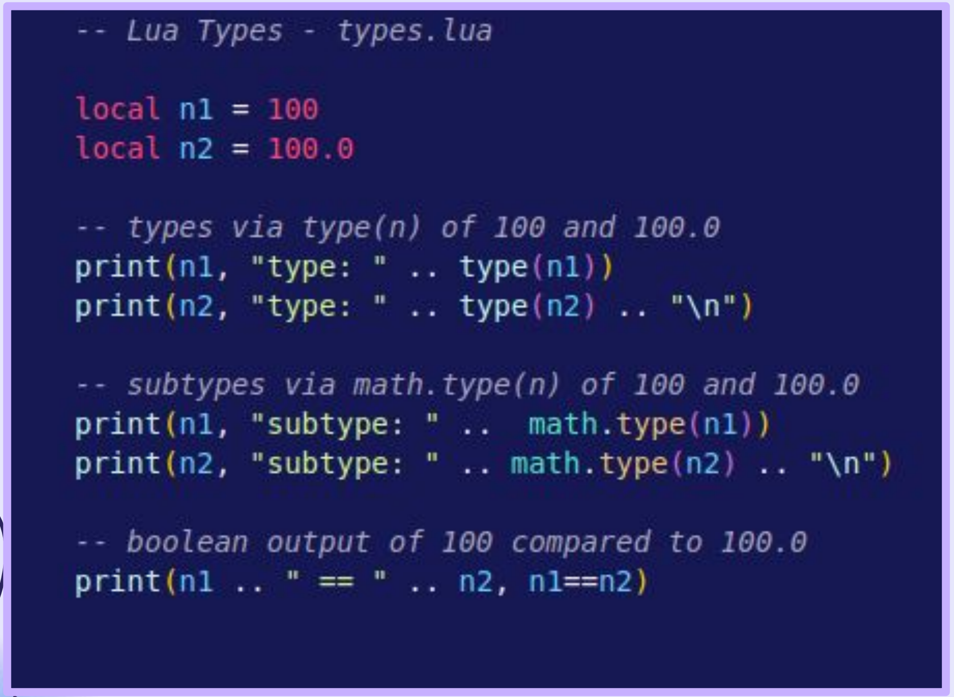


```lua
-- Lua Types - types.lua

-- Types
Num = 1
Bool = false
Nil = nil
Str = "string!"

-- Table (data struc)
Table = {"one", "two", "three"}

-- Additional
Userdata = io.stdin
Thread = coroutine.create(function() return 42 end)
Func = function(x) return x*100 end

Types = {Num, Bool, Nil, Str, Table, Userdata, Thread, Func}

print("Table elements with their types\n")
for i = 1, #Types do
    print(Types[i], "type == "  .. type(Types[i]) .. "\n")
end
```

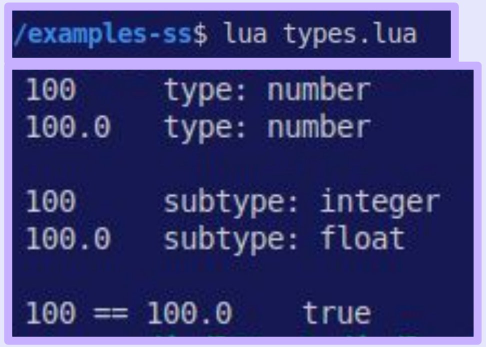


```
/examples-ss$ lua types.lua
Table elements with their types

1       type == number

false   type == boolean

string! type == string

nil     type == nil

table: 0x615cb0ddb670   type == table

file (0x722dba6038e0)   type == userdata

thread: 0x615cb0ddb728  type == thread

function: 0x615cb0ddba90        type == function
```

Code Example: ‘number’ Type

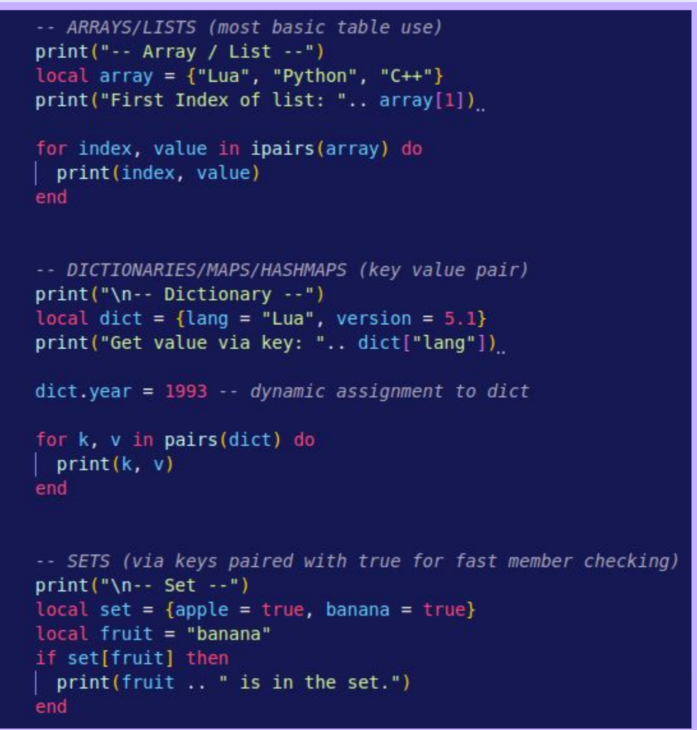


```lua
-- Lua Types - types.lua

local n1 = 100
local n2 = 100.0

-- types via type(n) of 100 and 100.0
print(n1, "type: " .. type(n1))
print(n2, "type: " .. type(n2) .. "\n")

-- subtypes via math.type(n) of 100 and 100.0
print(n1, "subtype: " ..  math.type(n1))
print(n2, "subtype: " .. math.type(n2) .. "\n")

-- boolean output of 100 compared to 100.0
print(n1 .. " == " .. n2, n1==n2)
```

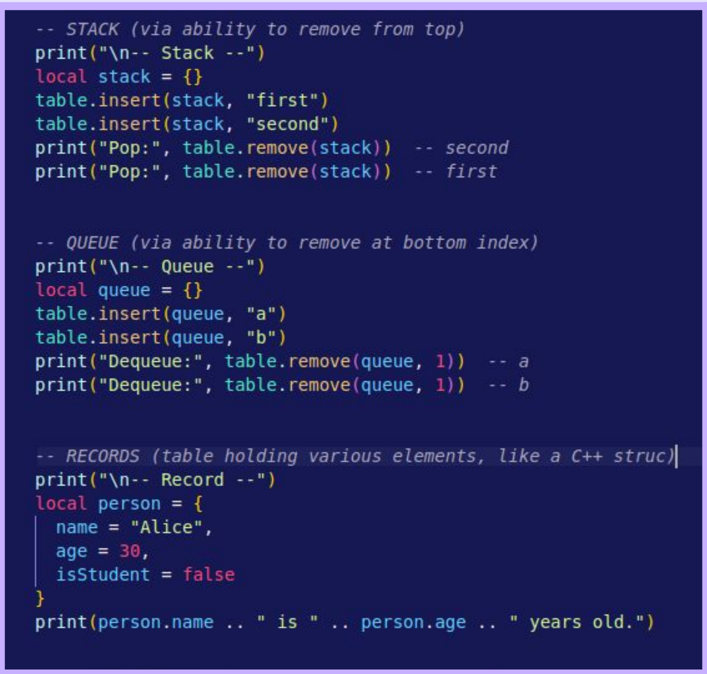


```
/examples-ss$ lua types.lua
100     type: number
100.0   type: number

100     subtype: integer
100.0   subtype: float

100 == 100.0    true
```

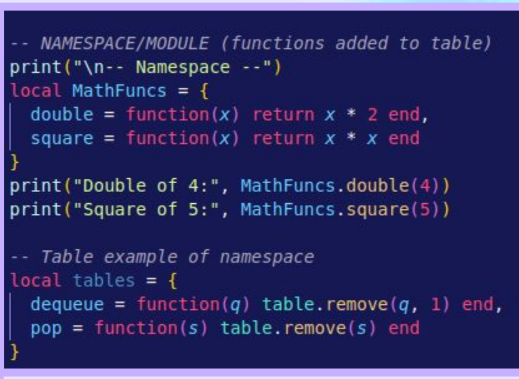
Code Example: Tables (Lua’s catch-all data struc)



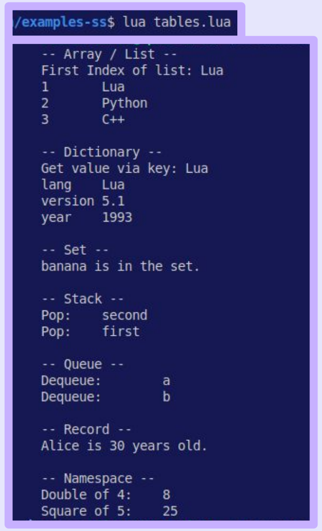
```lua
-- ARRAYS/LISTS (most basic table use)
print("-- Array / List --")
local array = {"Lua", "Python", "C++"}
print("First Index of list: ".. array[1])

for index, value in ipairs(array) do
  print(index, value)
end


-- DICTIONARIES/MAPS/HASHMAPS (key value pair)
print("\n-- Dictionary --")
local dict = {lang = "Lua", version = 5.1}
print("Get value via key: ".. dict["lang"])

dict.year = 1993 -- dynamic assignment to dict

for k, v in pairs(dict) do
  print(k, v)
end


-- SETS (via keys paired with true for fast member checking)
print("\n-- Set --")
local set = {apple = true, banana = true}
local fruit = "banana"
if set[fruit] then
  print(fruit .. " is in the set.")
end
```



```lua
-- STACK (via ability to remove from top)
print("\n-- Stack --")
local stack = {}
table.insert(stack, "first")
table.insert(stack, "second")
print("Pop:", table.remove(stack))  -- second
print("Pop:", table.remove(stack))  -- first


-- QUEUE (via ability to remove at bottom index)
print("\n-- Queue --")
local queue = {}
table.insert(queue, "a")
table.insert(queue, "b")
print("Dequeue:", table.remove(queue, 1))  -- a
print("Dequeue:", table.remove(queue, 1))  -- b


-- RECORDS (table holding various elements, like a C++ struc)
print("\n-- Record --")
local person = {
  name = "Alice",
  age = 30,
  isStudent = false
}
print(person.name .. " is " .. person.age .. " years old.")
```

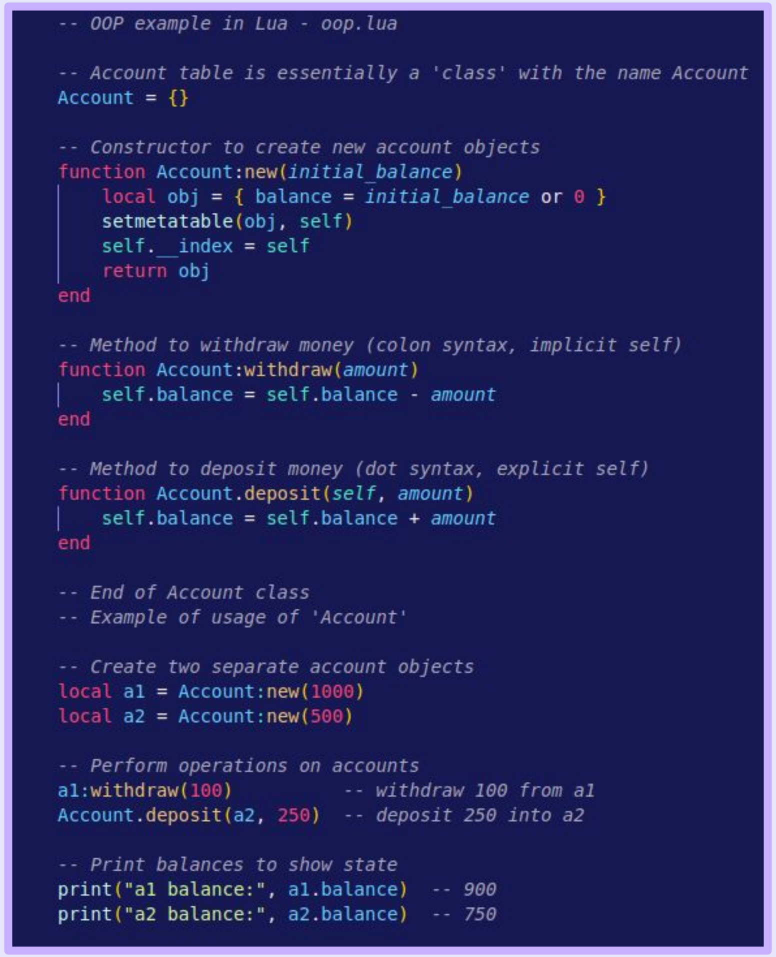


```lua
-- NAMESPACE/MODULE (functions added to table)
print("\n-- Namespace --")
local MathFuncs = {
  double = function(x) return x * 2 end,
  square = function(x) return x * x end
}
print("Double of 4:", MathFuncs.double(4))
print("Square of 5:", MathFuncs.square(5))

-- Table example of namespace
local tables = {
  dequeue = function(q) table.remove(q, 1) end,
  pop = function(s) table.remove(s) end
}
```



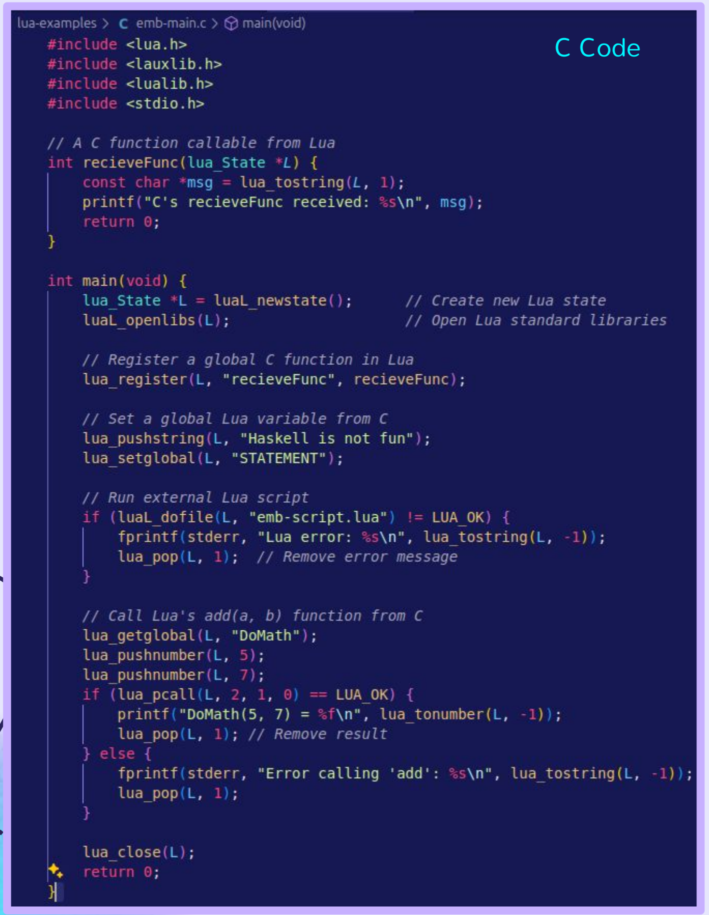
```
/examples-ss$ lua tables.lua
-- Array / List --
First Index of list: Lua
1       Lua
2       Python
3       C++

-- Dictionary --
Get value via key: Lua
lang    Lua
version 5.1
year    1993

-- Set --
banana is in the set.

-- Stack --
Pop:    second
Pop:    first

-- Queue --
Dequeue:        a
Dequeue:        b

-- Record --
Alice is 30 years old.

-- Namespace --
Double of 4:    8
Square of 5:    25
```

Code Example: OOP via tables and metatables

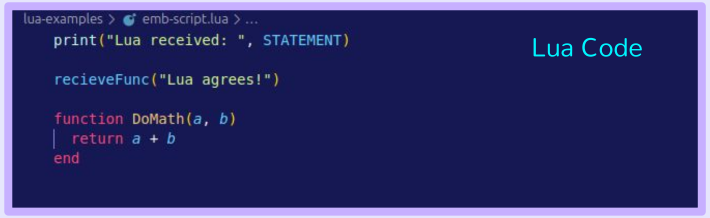


```lua
-- OOP example in Lua - oop.lua

-- Account table is essentially a 'class' with the name Account
Account = {}

-- Constructor to create new account objects
function Account:new(initial_balance)
    local obj = { balance = initial_balance or 0 }
    setmetatable(obj, self)
    self.__index = self
    return obj
end

-- Method to withdraw money (colon syntax, implicit self)
function Account:withdraw(amount)
    self.balance = self.balance - amount
end

-- Method to deposit money (dot syntax, explicit self)
function Account.deposit(self, amount)
    self.balance = self.balance + amount
end

-- End of Account class
-- Example of usage of 'Account'

-- Create two separate account objects
local a1 = Account:new(1000)
local a2 = Account:new(500)

-- Perform operations on accounts
a1:withdraw(100)          -- withdraw 100 from a1
Account.deposit(a2, 250)  -- deposit 250 into a2

-- Print balances to show state
print("a1 balance:", a1.balance)  -- 900
print("a2 balance:", a2.balance)  -- 750
```

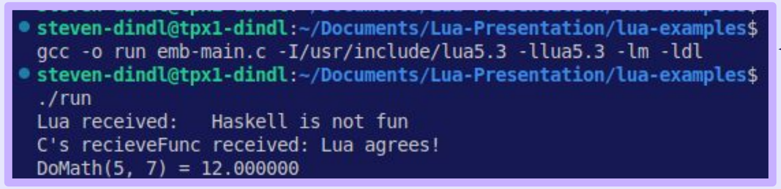
Code Example: C Embedding



```c
#include <lua.h>
#include <lauxlib.h>
#include <lualib.h>
#include <stdio.h>

// A C function callable from Lua
int recieveFunc(lua_State *L) {
    const char *msg = lua_tostring(L, 1);
    printf("C's recieveFunc received: %s\n", msg);
    return 0;
}

int main(void) {
    lua_State *L = luaL_newstate();      // Create new Lua state
    luaL_openlibs(L);                    // Open Lua standard libraries

    // Register a global C function in Lua
    lua_register(L, "recieveFunc", recieveFunc);

    // Set a global Lua variable from C
    lua_pushstring(L, "Haskell is not fun");
    lua_setglobal(L, "STATEMENT");

    // Run external Lua script
    if (luaL_dofile(L, "emb-script.lua") != LUA_OK) {
        fprintf(stderr, "Lua error: %s\n", lua_tostring(L, -1));
        lua_pop(L, 1);  // Remove error message
    }

    // Call Lua's add(a, b) function from C
    lua_getglobal(L, "DoMath");
    lua_pushnumber(L, 5);
    lua_pushnumber(L, 7);
    if (lua_pcall(L, 2, 1, 0) == LUA_OK) {
        printf("DoMath(5, 7) = %f\n", lua_tonumber(L, -1));
        lua_pop(L, 1); // Remove result
    } else {
        fprintf(stderr, "Error calling 'add': %s\n", lua_tostring(L, -1));
        lua_pop(L, 1);
    }

    lua_close(L);
    return 0;
}
```



```lua
print("Lua received: ", STATEMENT)

recieveFunc("Lua agrees!")

function DoMath(a, b)
  return a + b
end
```



```
steven-dindl@tpx1-dindl:~/Documents/Lua-Presentation/lua-examples$
gcc -o run emb-main.c -I/usr/include/lua5.3 -llua5.3 -lm -ldl
steven-dindl@tpx1-dindl:~/Documents/Lua-Presentation/lua-examples$
./run
Lua received:   Haskell is not fun
C's recieveFunc received: Lua agrees!
DoMath(5, 7) = 12.000000
```

Compile C

Output

*Had to install liblua5.3-dev

References

Slide 3: https://www.lua.org/history.html, https://en.wikipedia.org/wiki/Petrobras
Slide 5 & 10-14: https://www.lua.org/manual/5.3/
Slide 6+7: https://www.lua.org/versions.html,
https://stackoverflow.com/questions/27960235/what-is-a-tag-in-lua-4-0
Slide 8: http://lua-users.org/wiki/LuaUses, https://www.love2d.org/,
https://create.roblox.com/docs/luau
Slide 9: https://github.com/hengestone/lua-languages, https://en.wikipedia.org/wiki/Lua
Slide 11: https://www.lua.org/pil/2.3.html
Slide 14: https://lucasklassmann.com/blog/2019-02-02-embedding-lua-in-c/

General References: https://en.wikipedia.org/wiki/Lua, https://www.lua.org
Slides Template Credit: Slidesgo with elements from Flaticon and Freepix
Images: (Slide 3) https://www.nima.puc-rio.br/contato/
Logos belong to respective companies