Paradigma Imperativo

Delis Alexander Mejia García & Kenneth Leonardo Galdámez

Octubre 2024.

UNAH CAMPUS COPÁN.

Ingeniería En Sistemas.

Lenguajes de Programación.

Contenido

Que es Lua

Lua es un lenguage de programación extensible diseñado para una programación procedimental general con utilidades para la descripción de datos que gracias a su simplicidad facilita la programacion imperativa. También ofrece un buen soporte para la programación orientada a objetos y programación funcional. Se pretende que Lua sea usado como un lenguaje de script potente y ligero para cualquier programa que lo necesite. Lua está implementado como una biblioteca escrita en C limpio (esto es, en el subconjunto común de ANSI C y C++).

Siendo un lenguaje de extensión, Lua no tiene noción de programa principal (main): suele funcionar embebido en un cliente anfitrión, denominado programa contenedor o simplemente anfitrión (host). Éste puede invocar funciones para ejecutar un trozo de código Lua, puede escribir y leer variables de Lua y puede registrar funciones C para que sean llamadas por el código Lua. A través del uso de funciones C, Lua puede ser aumentado para abarcar un amplio rango de diferentes dominios, creando entonces lenguajes de programación personalizados que comparten el mismo marco sintáctico. La distribución de Lua incluye un programa anfitrión de muestra denominado lua, que usa la biblioteca de Lua para ofrecer un intérprete de Lua completo e independiente.

Lua es software libre, y se proporciona, como es usual, sin garantías, como se establece en su licencia. La implementación descrita en este manual está disponible en el sitio web oficial de Lua, www.lua.org.

Lua suele ser muy utilizado gracias a multitud de usos y ventajas que posee sobre muchos lenguajes, si se toma el tiempo de buscar los usos mas comunes de lua, encontrara que la mayoria de usos populares seran implemdntandolo en otros sistemas, usualmente de c, para facilitar la codificación de posibles modificaciones o ppñara no dar acceso al lenguaje original a usuarios exteriores, esos usos se dan debido a la agilidad que permite lua para trabajar y lo excepcionalmente rapido que es, simplificando el trabajo al equipo y a posibles programadores exteriores, ejemplos de estos usos spn por ejemplo robñox o cryengine.

Plataformas de desarrollo de videojuegos que permiten ingresar codigo de lua para scrypts por parte de los usuarios finales, world of warcraft y modificaciones de otros jiegos tambien aceptan addons de lua, varios projectos cerrados al publico tambien lo incorporan para facilitar cpdificar en el codigo base, o incluso se utiliza para dispositivos IoT pues lo portatil que es lo pone por encima de otras opciones, ademas de estar embebido en otros sistemas, por si solo es muy util ya que tiene una cantidad grande de frameworks que son muy populares, por ejemplo love2d para desarrollar videojuegos desde 0 usando lua como lenguaje base, o uno muy popular, torch que es de lps frameworks mas populares para inteligencia artificial, si se dedica a explorar prpyectos de lua encontrara muchps ejemplos de estos y otros grandes frameworks.

El Lenguaje.

Esta sección describe el léxico, la sintaxis y la semántica de Lua. En otras palabras, esta sección describe qué elementos (tokens) son válidos, cómo deben combinarse y qué significa su combinación.

Las construcciones del lenguaje se explicarán usando la notación BNF extendida usual, en la que {a} significa 0 o más aes, y [a] significa una a opcional. Los símbolos no terminales se muestran en itálica, las palabras clave (keywords) se muestran en negrita, y los otros símbolos terminales se muestran en un tipo de letra de paso fijo (typewriter), encerrada entre comillas simples. La sintaxis completa de Lua se encuentra al final de este manual

Convecciones léxicas

Identificadores:

Los identificadores (nombres de variables y campos) pueden contener letras, dígitos y guiones bajos (_), pero no pueden comenzar con un dígito.

Lua diferencia entre mayúsculas y minúsculas, por lo que and es una palabra reservada, pero And o AND son nombres válidos.

Nombres que comienzan con un guion bajo y letras mayúsculas (como _VERSION) están reservados para variables globales internas de Lua.

Palabras clave (reservadas y no pueden usarse como nombres):

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

Operadores y símbolos:

Lua utiliza símbolos como +, -, *, /, %, ^, # (operadores), y operadores de comparación como ==, ~=, <=, >=, <, >.

Otros símbolos incluyen paréntesis (), llaves {}, corchetes [], punto y coma, coma, punto., concatenación., y tres puntos ....

Strings:

Se pueden definir con comillas simples (') o dobles (").

Lua admite secuencias de escape similares a C, como \n para nueva línea, \t para tabulador, y \\ para una barra inversa.

Los strings largos pueden definirse con corchetes largos ([[...]]), y pueden abarcar varias líneas sin interpretar secuencias de escape.

Números:

Lua permite números enteros, decimales y notación científica, como 3, 3.14, 314.16e-2, 0xff (hexadecimal).

Comentarios:

Los comentarios de una línea comienzan con -- y llegan hasta el final de la línea.

Los comentarios largos comienzan con --[[y terminan con]], y pueden usarse para desactivar bloques de código.

valores y tipos

Lenguaje dinámicamente tipado:

Las variables no tienen tipos, solo los valores los tienen. No hay definiciones de tipos en el lenguaje.

Los valores almacenan su propio tipo y pueden ser almacenados en variables, pasados como argumentos o devueltos por funciones.

Tipos básicos:

nil: Representa la ausencia de valor. Es diferente de cualquier otro valor.

boolean: Tiene dos valores, true y false. Solo nil y false son considerados como falsos en condiciones.

number: Representa números reales en coma flotante (doble precisión).

string: Cadenas de caracteres que pueden contener cualquier valor de 8 bits, incluso el carácter cero (\0).

function: Lua puede manejar funciones escritas tanto en Lua como en C.

userdata: Permite almacenar datos arbitrarios en C que no tienen operaciones predefinidas en Lua.

thread: Representa procesos de ejecución, usados para implementar co-rutinas.

table: Implementa arrays asociativos, donde los índices pueden ser de cualquier tipo, excepto nil. Las tablas son el único mecanismo de estructuración de datos en Lua.

Tablas:

Las tablas pueden contener valores de cualquier tipo, excepto nil, y pueden ser heterogéneas.

Soportan arrays, tablas de símbolos, registros, grafos, árboles, entre otros.

Pueden almacenar funciones como métodos, lo que permite implementar objetos y comportamientos complejos, estas son parte de los atractivos mas grandes de lua, debido a la flexibilidad que permiteeztas tablas para trabajar, simplificando mucho el codigo en comparacion a otros lenguajes.

Referencias:

Los valores de tipo tablen, function, thread y userdata son objetos. Las variables que los contienen solo almacenan referencias a esos valores.

Las asignaciones y el paso de argumentos siempre manejan referencias, sin copiar los valores.

Función type:

La función de biblioteca type() devuelve un string con el tipo de un valor dado.

Variables

Tipos de variables:

Globales: Son las variables por defecto, a menos que se declaren explícitamente como locales.

Locales: Tienen un ámbito léxico, lo que significa que pueden ser accedidas dentro de las funciones definidas en el mismo ámbito.

Campos de tabla: Son las variables almacenadas dentro de una tabla, indexadas mediante corchetes o el operador punto.

Ámbito de las variables:

Las variables locales tienen un alcance limitado por el bloque o la función en la que se definen.

Las variables globales están disponibles en todo el código, a menos que sean sobreescritas por una variable local del mismo nombre.

Acceso a tablas:

Las tablas se indexan con corchetes, por ejemplo, t[i].

También se pueden acceder a los campos de una tabla usando t. nombre, equivalente a t["nombre"].

Metatablas:

El comportamiento del acceso a variables globales y campos de tabla puede ser modificado mediante metatablas, que definen eventos como gettable_event para personalizar cómo se acceden las variables.

Entornos:

Las variables globales se almacenan en una tabla de entorno asociada a cada función.

Las funciones pueden heredar el entorno de la función que las creó, y se puede manipular este entorno con getfenv y setfen

Generador de texto

```lua
-- Generador de texto

-- Listas de palabras
local sustantivos = {"gato", "perro", "niño", "profesor", "auto", "libro", "mujer", "hombre"}
local verbos = {"corre", "salta", "come", "duerme", "lee", "escribe", "canta", "habla"}
local adjetivos = {"rapido", "lento", "alto", "inteligente", "feliz", "triste", "grande", "pequeño"}
local adverbios = {"rápidamente", "cuidadosamente", "felizmente", "lentamente", "silenciosamente"}

-- Función para generar una palabra aleatoria de una lista
local function generar_aleatorio(lista)
    return lista[math.random(1, #lista)]
end

-- Función para generar una oración
local function generar_oracion()
    local sustantivo = generar_aleatorio(sustantivos)
    local verbo = generar_aleatorio(verbos)
    local adjetivo = generar_aleatorio(adjetivos)
    local adverbio = generar_aleatorio(adverbios)

    return "El " .. sustantivo .. " " .. verbo .. " " .. adverbio .. " y es " .. adjetivo .. "."
end

-- Generar 5 oraciones aleatorias
math.randomseed(os.time())  -- Inicializa el generador de números aleatorios
for i = 1, 5 do
    print(generar_oracion())
end
```

Cómo funciona:

El generador tiene listas de sustantivos, verbos, adjetivos y adverbios.

La función generar_aleatorio selecciona una palabra aleatoria de cada lista.

La función generar_oracion construye una oración simple en formato: "El [sustantivo] [verbo] [adverbio] y es [adjetivo]."

Se generan 5 oraciones aleatorias y se imprimen en la consola.

Tic tac toe de 2 jugadores

```lua
main.lua > [ ] wins
local wins = {
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9},
    {1, 4, 7},
    {2, 5, 8},
    {3, 6, 9},
    {1, 5, 9},
    {3, 5, 7}
}

local continue = true;
```

Se declara la tabla de posibles victorias, cada una es una lista con 3 valores semejantes a los del grid.

```lua
local continue = true;

while continue do
...

    local grid = {
        { "1", 0 },
        { "2", 0 },
        { "3", 0 },
        { "4", 0 },
        { "5", 0 },
        { "6", 0 },
        { "7", 0 },
        { "8", 0 },
        { "9", 0 }
    }
...
...
...
    local check = true
    local userplay = 0
    local u = 2
    local full = false

    --LOOP DE TURNOS~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    while check do
```

Se inicia el while principal, cada iteración inicia una partida nueva, reiniciando los valores del grid que es una tabla que simboliza el tqblrero de juego, cada parte es una casilla con un valor y una variable indicando que jugador la tomo, 0 si esta vacia, la variable check es para finalizar el segundo bucle, u indica que jugador juega, userplay es para guardar la casilla que selecciono y full por si se llena el grid.

```lua
while check do
    --.....................................................
    for i = 1, 255 do
        print()
    end
    full = false

--SELECCION DE TURNO.....................................................
    if u == 1 then
        u = 2
    else
        u = 1
    end

--IMPRESION DE GRID.....................................................
    print(grid[1][1] .. " | " .. grid[2][1] .. " | " .. grid[3][1])
    print("--|---|--")
    print(grid[4][1] .. " | " .. grid[5][1] .. " | " .. grid[6][1])
    print("--|---|--")
    print(grid[7][1] .. " | " .. grid[8][1] .. " | " .. grid[9][1])

--SELECCION DE CASILLA.....................................................
    print("jugador " .. u .. ", Indique posicion para marcar (1-9): ")
```

El segundo bucle inicia los turnos, en el primer for, se despeja la pantalla y el if es para alternar a u, iniciara con 1 para jugador 1, los print dibujan el grid, cada casilla con su valor que mostrar.

```lua
69
70      --REGISTRO EN EL GRID.....................
71          grid[userplay][2] = u
72
73          if u == 1 then
74              grid[userplay][1] = "\27[31mX\27[0m"
75          else
76              grid[userplay][1] = "\27[34m0\27[0m"
77          end
78
79      --CHECKEO DE VICTORIA.....................
80          for _, winCombination in ipairs(wins) do
81              if  grid[winCombination[1]][2] == u and grid[winCombination[2]][2] == u and
82                  grid[winCombination[3]][2] == u then
83
84                  print("Jugador " .. u .. " gana!")
85                  print("Presiona 'Enter' para continuar...")
86                  io.read()
87                  check = false
88                  break
89              end
90          end
91
92          for i = 1, 9 do
```

Se evalua si lo ingresado por el usuario ingreso valores validos con los if, checando que el segundo valor de la casilla seleccionada sea 0, y que la casilla en si exista.

Si existe, se cambia el primer valor de la casilla por x o 0 dependiendo del valor de u y por tanto el jugador, y su segundo valor pasa de 0 a el de u.

También se valida una posible victoria con el bucle for, iterando por win para saber si los valores de las casillas de grid indicadas por win encajan las 3 con el jugador de ese turno, ósea u, si hay victoria, se indica y se corta el segundo bucle al cambiar el valor de check, para empezar otra iteración del bucle exterior y por tanto una nueva partida.

```
        end

        for i = 1, 9 do
            if grid[i][2] == 0 then
                full = true
            end
        end

        if  not full then
            print("Se lleno el campo...")
            print("Presiona 'Enter' para continuar...")
            io.read()
            check = false
        end
    ----------------------------------------
    end
    ----------------------------------------------------------------------------
...
end
```

Al final, de no haber victoria se evalua si el campo esta lleno con un buvle que revisa cada casilla del grid hasta encontrar al menos un posible valor 0, de no haberlo lo indica y termina la partida.

Bibliografía

Lua. (s.f.). *Lua*. Obtenido de https://www.lua.org/manual/5.1/es/manual.html

Link de video

https://youtu.be/zR9eYWCmObA?si=W_OKpPfkMhv6ODNe