

Introducción a Lua

Master Programación Videojuegos
Scripting

• © Diego Garcés 12/04/2016 •

Estructuras de control

...
Condicionales y Bucles

• © Diego Garcés 12/04/2016 •

Condicionales

- Necesaria una condicion
 - falso: false o nil
 - cierto: cualquier otra cosa

Estructura

- if condición then
 instrucciones
end

```
muerto = true
if muerto then
    print("Has perdido todas las vidas")
    print("Game Over")
end
```

• © Diego Garcés 12/04/2016 •

Condicionales (else)

Estructura

- if condición then
 instrucciones
else
 instrucciones
end

```
vidas = 5
-- codigo
if vidas <= 0 then
    print("Has perdido todas las vidas")
    print("Game Over")
else
    print("Todavía te quedan " .. vidas ..
        " vidas")
end
```

Ejemplo

```
a = 5
b = 7
op = "+"
--[
op = "-"
op = "*"
op = "/"
--]]
-- Código para hacer la operación adecuada
print(c)
```

Ejemplo

```
if op == "+" then
    c = a + b
else if op == "-" then
    c = a - b
else if op == "*" then
    c = a * b
else if op == "/" then
    c = a / b
end
end
end
```

Condicionales (elseif)

- Estructura**
- **if** condición **then**
instrucciones
 - **elseif** otra_condición **then**
instrucciones
 - **else**
instrucciones
 - **end**
- else es opcional

Ejemplo

```
a = 5
b = 7
op = "+"
--[
op = "-"
op = "*"
op = "/"
--]]
-- Codigo para hacer la operacion adecuada
print(c)
```

Ejemplo

```
if op == "+" then
  c = a + b
elseif op == "-" then
  c = a - b
elseif op == "*" then
  c = a * b
elseif op == "/" then
  c = a / b
else
  print("Operador desconocido")
end
```

Bucles (while)

Estructura

- **while** condición **do**
 instrucciones
end
- Mientras la condición sea cierta se repiten las instrucciones
 - Si es falsa al principio, no se ejecutan nunca
 - Si nunca es falsa, tenemos un bucle infinito

Ejemplo

```
numero = 10
-- calculo de factorial
print(factorial)
```

Ejemplo

```
numero = 10
factorial = 1
while numero > 1 do
    factorial = factorial * numero
    numero = numero - 1
end
print(factorial)
```

Bucles (repeat)

- Inverso al while
- **Estructura**
- **repeat**
 - instrucciones
- **until** condición
- Ejecuta instrucciones mientras condición es falsa
 - Hasta que la condición es cierta
 - Si la condición cierta de inicio, se ejecuta instrucciones 1 vez
 - Si la condición nunca es cierta, tenemos un bucle infinito

Ejemplo

```
numero = 10
-- calculo de factorial
print(factorial)
```

Ejemplo

```
numero = 10
factorial = 1
if numero > 1 then
  repeat
    factorial = factorial * numero
    numero = numero - 1
  until numero <= 1
end
print(factorial)
```

Bucles (for)

Estructura

- **for** var = begin, end, step **do**
instrucciones
end
- Var va tomando valores
 - desde begin, hasta end
 - incrementando step en cada iteración
- Step es opcional
 - si no se especifica, se usa 1 como valor por defecto

Bucles (for)

A tener en cuenta

- La expresión end no es una condición
 - Es un valor numérico
- Las expresiones begin, end y step sólo se evalúan una vez
 - Si variables que usan, cambian su valor en el bucle, no se ven afectadas
- La variable "var" sólo es válida en el bucle
- No es posible cambiar el valor de "var"
 - Resultados no definidos

Ejemplo

```
numero = 10
-- calculo de factorial
print(factorial)
```

Ejemplo

```
numero = 10
factorial = 1
for num = 2, numero do
    factorial = factorial * num
end
print(factorial)
```

```
numero = 10
factorial = 1
for num = numero, 2, -1 do
    factorial = factorial * num
end
print(factorial)
```

Bucles (break)

- Instrucción para salir de un bucle
- Si se cumplen ciertas condiciones → terminar bucle
 - Cambiar la variable de control para cumplir la condición del bucle
 - Usar break
- Sólo se puede usar dentro de un bucle
 - No se puede usar dentro de una condición para saltar ciertas instrucciones

Bucles (break)

```
intentos = 5
while intentos > 1 do
    intentos = intentos - 1
    -- pedir elegir una carta
    if carta == "5 bastos" then
        break
    elseif carta == "As de oros" then
        break;
    else
        -- Hacer algo con la carta
    end
end
end
```

Estructuras de Datos

...

Arrays

© Diego Garcés

12/04/2016

Arrays

- Utilizan la estructura genérica: tabla
- No tienen tamaño fijo
 - Crecen y decrecen dinámicamente
- No se puede copiar un array directamente
 - Las variables arrays son sólo referencias a los datos
 - Si se asignan, se asigna la referencia → estamos accediendo a los mismos datos
- No se pueden comparar los contenidos de dos arrays directamente
 - Dos arrays sólo son iguales si son el mismo objeto

© Diego Garcés

12/04/2016

Arrays

Estructura

- array = {dato, dato, ...}
- array[indice] = valor
- var = array[indice]

```
creatures = {}
creatures[1] = "grunt"
creatures[2] = "goblin"
print(creatures[2])
```

```
creatures = {"grunt", "goblin"}
print(creatures[2])
```

© Diego Garcés

12/04/2016

Arrays

- Si no existe un índice, su valor es nil
 - Igual que una variable no inicializada
- Para obtener el número de elementos
 - Operador longitud: #

```
creatures = {"grunt", "goblin", "mage", "dragon"}
print(creatures[10])
num_creatures = #creatures
print(#creatures[2])
print(#creatures[#creatures])
```

Arrays

Insertar elementos

- Asignar un valor a un índice
 - Salvo que sea al final, no estamos insertando
- **table.insert**(array, posición, valor)

Borrar elementos

- Poner a nil un índice
 - Salvo que sea al final, no cambia la longitud del array
- **table.remove**(array, posición)
 - Se compactan los datos del array → se eliminan todos los nil intermedios

Ejemplo

```
creatures = {"grunt", "goblin", "mage", "dragon"}
creatures[5] = "kobold"
print(#creatures)
table.insert(creatures, 3, "warlock")
print(#creatures, creatures[3])

creatures[6] = nil
print(#creatures)
table.remove(creatures, 1)
print(#creatures, creatures[1])
```

Ejemplo

```
creatures = {"grunt", "goblin", "mage", "dragon"}
creatures[3] = nil
print(#creatures)
table.remove(creatures, 1)
print(#creatures, creatures[1])
```

Funciones

...

Declaración

Estructura

- **function** <nombre>([nombre_parámetro, ...])
instrucciones
end
- No se especifica qué tipo de dato se devuelve
 - No tiene por qué devolver siempre el mismo tipo
- Los parámetros son opcionales
 - Siempre hay que poner los paréntesis

Ejemplo

```
function factorial(numero)
  -- instrucciones
end
```

```
function suma(op1, op2)
  -- instrucciones
end
```

```
function print_gameOver()
  print("Game Over")
end
```

```
function get_name(personaje)
  -- devolver nombre personaje
end
```

Llamada

- Estructura**
- resultado = <nombre>([nombre_parametro, ...])
 - Siempre hay que poner los paréntesis
 - Aunque no haya parámetros

```
numero = 10
res_factorial = factorial(numero)
print(res_factorial)
print(factorial(4))
```

Parámetros

- Funcionan igual que la asignación múltiple
- Si se mandan menos → rellena con nil
- Si se mandan más → descarta los últimos

```
function choose(a, b)
  -- instrucciones
end
```

Llamada	Parámetros
choose(3)	a = , b =
choose(3, 4)	a = , b =
choose(3, 4, 5)	a = , b =

Parámetros

- Funcionan igual que la asignación múltiple
- Si se mandan menos → rellena con nil
- Si se mandan más → descarta los últimos

```
function choose(a, b)
  -- instrucciones
end
```

Llamada	Parámetros
choose(3)	a = 3, b = nil
choose(3, 4)	a = 3, b = 4
choose(3, 4, 5)	a = 3, b = 4 (5 descartado)

Parámetros

- No se especifica el tipo de los parámetros
 - Se puede mandar cualquier tipo
- No se chequea que el valor sea del tipo correcto
- Error si hacemos una operación no permitida con ese dato
 - Al ejecutar la instrucción inválida
 - Al cargar el script no dará error
- Ayudarse con nomenclatura
- Ayudarse con comentarios

Parámetros

```
function run(a, b)
  c = a + b
end
```

```
add(5, "Nathan")
```

```
function add(a, b)
  c = a + b
end
```

```
function add(a, b)
  -- a and b must be numbers
  c = a + b
end
```

```
function add(number1, number2)
  c = number1 + number2
end
```

Return

- Una función puede devolver más de un valor
- Estructura**
- `return` valor, [valor, ...]
 - Termina la ejecución de la función
 - Si no se incluye
 - Se devuelve nil al final de la función

```
function add(number1, number2)
  return number1 + number2
end
```

Ejemplos

```
function add(number1, number2)
  print("Adding", number1, number2)
  return number1 + number2
end
```

```
function escribe()
  print("Hello World")
end
res = escribe()
print(res)
```

```
res = add(5, 7)
add(5, 7)
```

```
function add(number1, number2)
  print("Adding", number1, number2)
  return number1 + number2
  if number1 > number2 then
    print("Number1 is higher")
  end
end
```

Return Múltiple

- Se pueden devolver varios valores con un return
- No tienen que ser del mismo tipo

```
function modify(b1, b2)
  -- b1 and b2 must be numbers
  modB1 = b1 + 1
  modB2 = b2 + 1
  return modB1, modB2
end
```

```
function modify(b1, b2)
  -- b1 and b2 must be numbers
  if needToModify() then
    modB1 = b1 + 1
    modB2 = b2 + 1
    return "modified", modB1, modB2
  else
    return "unmodified"
  end
end
```

Return Múltiple

- Se pueden devolver varios valores con un return
- No tienen que ser del mismo tipo

```
function modify(b1, b2)
  -- b1 and b2 must be numbers
  modB1 = b1 + 1
  modB2 = b2 + 1
  return modB1, modB2
end
```

```
function modify(b1, b2)
  -- b1 and b2 must be numbers
  if needToModify() then
    modB1 = b1 + 1
    modB2 = b2 + 1
    return modB1, modB2, "modified"
  else
    return "unmodified"
  end
end
```

© Diego Garcés12/04/2016

Return Múltiple

- Los valores se recogen igual que una asignación

```
function modify(b1, b2)
  -- b1 and b2 must be numbers
  modB1 = b1 + 1
  modB2 = b2 + 1
  return modB1, modB2
end
```

```
newB1, newB2 = modify(4, 8)
```

© Diego Garcés12/04/2016

Return Múltiple

Reglas

- Las mismas que en asignación múltiple
 - Si faltan se rellena con nil
 - Si sobran se descartan
- Si se usa en una operación (expresión)
 - Se usa sólo el primer valor devuelto
- Si se usa al final de una lista de expresiones
 - Se usan todos los valores
- Si se usa al principio o en medio
 - Se usa sólo el primer valor devuelto

© Diego Garcés12/04/2016

Return Múltiple

- Si faltan se rellena con nil

```
a, b, c = modify(3, 9)
print(c) -- nil
```
- Si sobran se descartan

```
newNumber = modify(4, 8)
print(newNumber) -- 5
```

```
function modify(b1, b2)
  -- b1 and b2 must be numbers
  modB1 = b1 + 1
  modB2 = b2 + 1
  return modB1, modB2
end
```

© Diego Garcés

12/04/2016

Return Múltiple

- Si se usa en una operación (expresión)
 - Se usa sólo el primer valor devuelto

```
c = 5 + modify(3, 9)
print(c) -- 9 = 5 + (3 + 1)
```

```
function modify(b1, b2)
  -- b1 and b2 must be numbers
  modB1 = b1 + 1
  modB2 = b2 + 1
  return modB1, modB2
end
```

© Diego Garcés

12/04/2016

Return Múltiple

- Si se usa al final de una lista de expresiones
 - Se usan todos los valores

```
a, b = modify(1, 5)
print(a, b) -- 2, 6
a, b, c = 4, modify(1, 5)
print(a, b, c) -- 4, 2, 6
```

```
function other(a1, a2)
  print(a1, a2)
end

other(modify(5, 15))
```

```
function modify(b1, b2)
  -- b1 and b2 must be numbers
  modB1 = b1 + 1
  modB2 = b2 + 1
  return modB1, modB2
end

function other(a1, a2, a3)
  print(a1, a2, a3)
end

other(25, modify(5, 15))
```

© Diego Garcés

12/04/2016

Return Múltiple

- Si se usa al principio o en medio
 - Se usa sólo el primer valor devuelto

```
a, b = modify(1, 5)
print(a, b) -- a <- 1, b <- 5
a, b, c = modify(1, 5), 4
print(a, b, c) -- a <- 1, b <- 5, c <- 4
```

```
function other(a1, a2, a3, a4)
  print(a1, a2, a3, a4)
end
other(5, modify(5, 15), 9)
-- a1 <- 5, a2 <- 16, a3 <- 9, a4 <- nil
```

```
function modify(b1, b2)
  -- b1 and b2 must be numbers
  modB1 = b1 + 1
  modB2 = b2 + 1
  return modB1, modB2
end
```

```
function other(a1, a2, a3)
  print(a1, a2, a3)
end
other(25, modify(5, 15))
-- a1 <- 26, a2 <- 16, a3 <- nil
```

© Diego Garcés

12/04/2016

Return Múltiple

- Si se usa al principio o en medio
 - Se usa sólo el primer valor devuelto

```
a, b = modify(1, 5)
print(a, b) -- a <- 2, b <- 6
a, b, c = modify(1, 5), 4
print(a, b, c) -- a <- 2, b <- 6, c <- nil
```

```
function other(a1, a2, a3, a4)
  print(a1, a2, a3, a4)
end
other(5, modify(5, 15), 9)
-- a1 <- 5, a2 <- 6, a3 <- 9, a4 <- nil
```

```
function modify(b1, b2)
  -- b1 and b2 must be numbers
  modB1 = b1 + 1
  modB2 = b2 + 1
  return modB1, modB2
end
```

```
function other(a1, a2, a3)
  print(a1, a2, a3)
end
other(25, modify(5, 15))
-- a1 <- 26, a2 <- 6, a3 <- nil
```

© Diego Garcés

12/04/2016
