Introducción a Lua

Master Programación Videojuegos Scripting

© Diego Garcés

2/04/2016

Estructuras de control Condicionales y Bucles

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

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("Todavia te quedan " ... vidas ... " vidas") end

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

```
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
    end
end
```

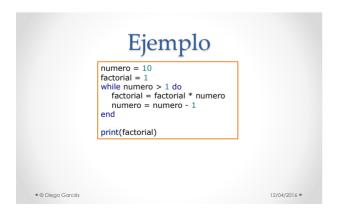
Condicion	ales (elseif)
<u>Estructura</u>	 else es opcional
• if condición then	
instrucciones	
elseif otra_condición th	en
instrucciones	
else	
instrucciones	
end	
	12/04/2016 ●


```
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
```

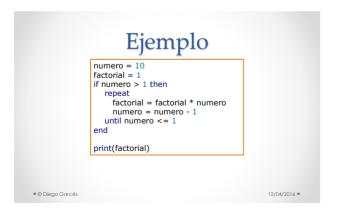
Bucles (while) Estructura • while condición do instrucciones end • Mientras la condición sea cierta se repiten las instrucciones • Si es falsa al principio, no se ejecutan nuna • Si nunca es falsa, tenemos un bucle infinito

	Ejemplo	
	numero = 10	
	calculo de factorial	
	print(factorial)	
• © Diego Garcés		12/04/2016 ●



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) ◆ © Diego Garcés



Bucles (for)

Estructura

• for var = begin, end, step do instrucciones

end

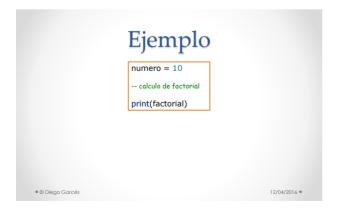
- Var va tomando valores

 - o desde begin, hasta endo incrementando step en cada iteración
- Step es opcional
 - o si no se especifica, se usa 1 como valor por defecto

Bucles (for)

A tener en cuenta

- · La expresión end no es una condición
 - o Es un valor numérico
- Las expresiones begin, end y step sólo se evalúan <u>una vez</u>
 - o 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"
- o Resultados no definidos



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)

0	Instrucción para salir de un bucle
	Si se cumplen ciertas condiciones → terminar bucl Cambiar-la variable de control para cumplir-la condición del bucle 0 <u>Usar break</u>
•	Sólo se puede usar dentro de un bucle o 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 carrta
if carta == "5 bastos" then
break
elseif carta == "As de oros" then
break;
else
-- Hacer algo con la carrta
end
end

*© Diego Garcés
```

Estructuras de Datos Arrays	
• © Diego Garcés	12/04/2016 ●

Arrays

- Utilizan la estructura genérica: tabla
- No tienen tamaño fijo
- o 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
 - o Dos arrays sólo son iguales si son el mismo objeto

© Diego Garce

2/04/2016

Arrays Estructura a gray = {dato, dato, ...} a gray[indice] = valor var = array[indice] creatures = {} creatures[1] = "grunt" creatures[2] = "qoblin" print(creatures[2]) creatures = {"grunt", "goblin"} print(creatures[2])

Arrays • Si no existe un índice, su valor es nil o Igual que una variable no inicializada Para obtener el número de elementos o Operador longitud: # creatures = {"grunt", "goblin", "mage", "dragon"} print(creatures[10]) num_creatures = #creatures print(#creatures[2]) print(#creatures[#creatures])

© Diego Garcés

Borrar elementos

© Diego Garcés

Arrays Insertar elementos · Asignar un valor a un índice o Salvo que sea al final, no estamos insertando • table.insert(array, posición, valor) · Poner a nil un índice o Salvo que sea al final, no cambia la longitud del array • table.remove(array, posición)

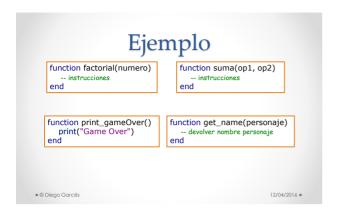
Ejemplo creatures = {"grunt", "goblin", "mage", "dragon"} creatures[5] = "kobold" print(#creatures) table.insert(creatures, 3, "warlock") print(#creatures, creatures[3]) creatures[6] = nil print(#creatures) table.remove(creatures, 1) print(#creatures, creatures[1]) • © Diego Garcés

Se compactan los datos del array → se eliminan todos los nil intermedios

creatures = { creatures[3] = print(#creatures[3])	res)	ayun }
	res, creatures[1])	

	Funciones	
• © Diego Garcés		12/04/2016 •

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

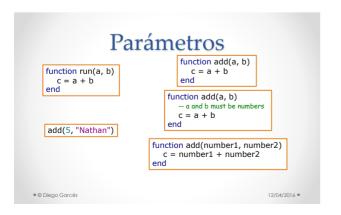


	Llamada	
 Siempre 	o = <nombre>([nombre_pardhay que poner los paréntesi</nombre>	
	numero = 10 res_factorial = factorial(numero) print(res_factorial) print(factorial(4))	

Pará	metros	
 Funcionan igual que la Si se mandan menos → Si se mandan más → d 	rellena con nil	
function choose(instrucciones end	(a, b)	
Llamada	Parámetros	
choose(3)	a = , b =	
choose(3, 4)	a = , b =	
choose(3, 4, 5)	a = , b =	
© Diego Garcés	1	12/04/2016 •

Parái	metros	
 Funcionan igual que la Si se mandan menos → Si se mandan más → de 	rellena con nil	
function choose(instrucciones end	a, b)	
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)	
• © Diego Garcés		12/04/2016 •

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



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

function add(number1, number2) print("Adding", number1, number2) return number1 + number2 end res = add(5, 7) add(5, 7) function add(number1, number2) print("Adding", 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 nedeToModify() then modB1 = b1 + 1 modB2 = b2 + 1 return modified", modB1, modB2 else return "unmodified" end 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 .cae1, modE return "unmodified" end end return modB1, modB2, "modified"

Return Múltiple

· Los valores se recogen igual que una asignación

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

newB1, newB2 = modify(4, 8)

Return Múltiple

Reglas

- · Las mismas que en asignación múltiple
 - o Si faltan se rellena con nil
 - o Si sobran se descartan
- Si se usa en una operación (expresión)
 - o 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
 - o Se usa sólo el primer valor devuelto

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 **Dlego Garcés* function modify(b1, b2) -- b1 and b2 must be numbers modB1 = b1 + 1 nodB2 = b2 + 1 return modB1, modB2 end

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

```
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))

• © Diego Garcés

• Si se usa al final de una lista de expresiones

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))
```

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 <- , b <- a, b, c = modify(1, 5), 4 print(a, b, c) -- a <- , b <- , c < 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, a4) print(a1, a2, a3, a4) end other(5, modify(5, 15), 9) -- a1 <- , a2 <- , a3 <- , a4 < **P Dieao Garcés* function other(a1, a2, a3) end other(25, modify(5, 15)) -- a1 <- , a2 <- , a3 <-

Return N • Si se usa al principio o en n	*			
se usa sólo el primer valor devuelto a, b = modify(1, 5) print(a, b) α <- 2, b <- 6 a, b, c = modify(1, 5), 4 print(a, b, c) α <- 2, b <- 4, c <- 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, a4) print(a1, a2, a3, a4) end other(5. modifv(5. 15). 9) a1 <- 5, a2 <- 6, a3 <- 9, a4 <- nil	function other(a1, a2, a3) print(a1, a2, a3) end other(25. modifv(5. 15)) a1 <- 25, a2 <- 6, a3 <- nil			
● © Diego Garcés	12/04/201	6 ●		