Compiladores e paradigmas da programação | 15/09/2025

Ciência da Computação

Lua

Linguagem de programação brasileira, de código aberto e multiplataforma

INTEGRANTES

Alicia Monteiro, Eduardo Couto, João Vitor Fernandes, Kleiton Josivan e Robert Danilo

SUMÁRIO

TIPOS DE DADOS SUPORTADOS

Existem **oito tipos básicos** em Lua:

Nil: Ausência de valor útil
Boolean: Valores true e false
Number: Números de ponto flutuante com dupla precisão
String: Sequência imutável de bytes
Function: Funções escritas em Lua ou C
Userdata: Dados arbitrários de C
Thread: Threads independentes (corrotinas)
Table: Arrays associativos

TIPOS DE DADOS SUPORTADOS

Exemplos:

```
print(type("Hello world"))    --> string
print(type(10.4*3))           --> number
print(type(print))            --> function
print(type(type))             --> function
print(type(true))             --> boolean
print(type(nil))              --> nil
print(type(type(X)))          --> string
```

TIPOS DE DADOS SUPORTADOS

Exemplos:

```
-- Tipo table

local minha_tabela =
{
 nome = "Lua",
 ano = 1993,
 10, 20, 30
}
print("Tipo table:", type(minha_tabela))

print("Acessando tabela:", minha_tabela.nome,
minha_tabela[1])
```

IMPLEMENTAÇÃO DOS TIPOS

Tipo	Detalhes de Implementação
number	• Double-precision floating-point (64 bits, IEEE 754) • Representa inteiros até 10^15 sem erros de arredondamento • Pode ser compilado para usar inteiros de 32/64 bits
string	• Imutáveis, com cabeçalho de 16 bytes + comprimento • 8-bit clean (pode conter qualquer byte, incluindo zeros) • Internadas em um pool global para economia de memória
table	• Estrutura híbrida: array + hash table • Arrays são dinâmicos, sem tamanho fixo • Índices podem ser qualquer valor exceto nil e NaN • Otimização automática para arrays sequenciais

PONTEIROS E REFERÊNCIAS

Passagem por Referência em Lua

- Lua não possui ponteiros explícitos como C.
- Objetos como **table, string e function** são manipulados por referência.
- Garbage collector gerencia a memória automaticamente.
- Benefício: evita erros comuns com ponteiros inválidos.

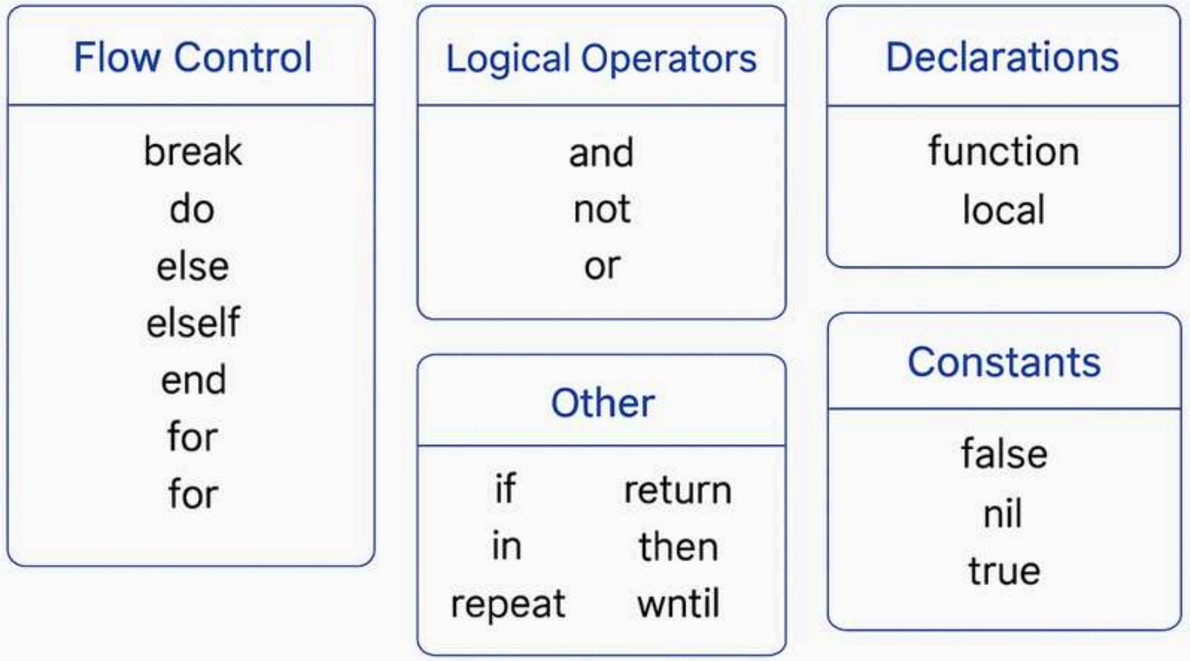
```
Local a = {x = 1,y = 2}
Local b = a
b[x] = 99
print(a[x]) -- saída: 99
```

PALAVRAS-CHAVE E RESERVADAS

- Lua é case-sensitive: **and ≠ And ≠ AND**
- Nomes de variáveis, campos e rótulos podem conter letras, dígitos e _, mas não podem começar com dígito.
- Não podem ser palavras reservadas.

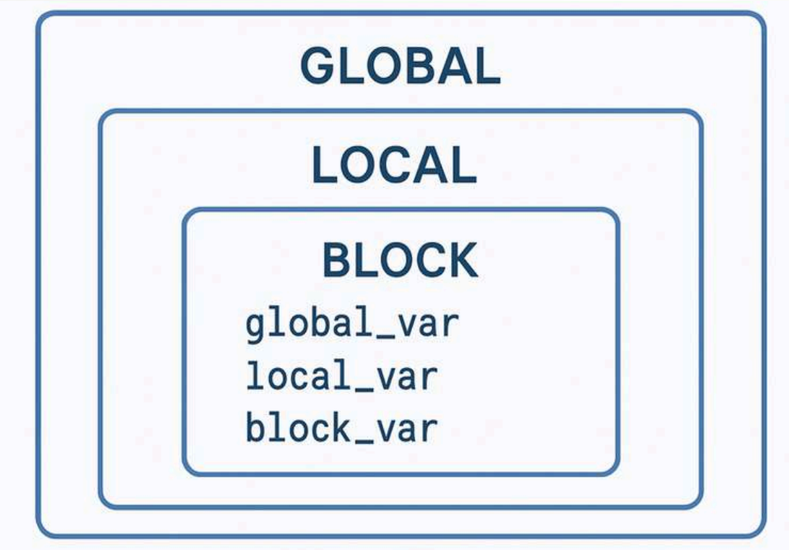
- Convenção: evitar nomes que começam com _ seguido de letras maiúsculas (ex.: _VERSION).

As seguintes palavras-chave são reservadas e não podem ser usadas como nomes:



VARIÁVEIS

- Existem três tipos de variáveis em Lua: **globais, locais,** campos de **tabela**
- Valor inicial de qualquer variável é nil
- Escopo léxico: variáveis locais são acessíveis por funções definidas dentro do escopo



Vinculação	Variáveis são globais por padrão, a menos que declaradas como **LOCAL**
Case-sensitive	**nome, Nome e NOME** são variáveis diferentes
Escopo	• Variáveis locais: do ponto de declaração até o fim do bloco • Variáveis globais: visíveis em todo o programa
Tempo de vida	• Locais: enquanto o escopo estiver ativo • Globais: durante toda a execução do programa
Sintaxe especial	**var.nome** é açúcar sintático para **var["nome"]**

LUA SUPORTA DIVERSOS TIPOS DE OPERADORES:

Operadores binários são associativos à esquerda, exceto ^ e .., que são associativos à direita.

Aritmétcios	• Binários: + (adição), - (subtração), * (multiplicação), / (divisão) • Unário: - (negação) • Exponenciação: ^
Relacionais	• **< > <= >= == ~=** • Todos retornam boolean (true ou false).
Lógico	• And, or e not
Concatenação	• Operador: .. (dois pontos) • Se algum operando for número, Lua converte automaticamente para string.

PRECEDÊNCIA DE OPERADORES EM LUA

Do maior para o menor:

1. **^ (exponenciação)**
2. **not, - (unário)**
3. ***, /**
4. **+, -**
5. **.. (concatenação de strings)**
6. **<, >, <=, >=, ~=, ==**
7. **and**
8. **or**

ESTRUTURAS DE CONTROLE

Lua oferece um conjunto convencional de estruturas de controle, similares às de outras linguagens de programação:

Estruturas condicionais

if-then-else: Executa blocos de código com base em condições

Estruturas de repetição

- **while**: Executa enquanto uma condição for verdadeira
- **repeat-until**: Executa pelo menos uma vez até uma condição ser verdadeira
- **for**: Numérico (com contador) ou genérico (com iterador)

Desvios de fluxo

- **break:** Termina a execução do loop mais interno
- **goto:** Transfere o controle para um rótulo
- **return**: Retorna valores de uma função

ESTRUTURAS DE CONTROLE

Exemplos:

```
-- Exemplo de if-then-elseif-else

local nota = 85
local resultado

if nota >= 90 then
 resultado = "Conceito A"
elseif nota >= 80 then
 resultado = "Conceito B"
elseif nota >= 70 then
 resultado = "Conceito C"
```

```
-- Exemplo de for numérico
for i = 1, 5 do
print("Valor de i:", i)
end
```

```
-- Exemplo de while
local contador = 1
while     contador     <=     5
doprint("Contador:", contador)
    contador = contador + 1
end
```

```
-- Exemplo de for genérico
local  frutas  =  {"maçã",  "banana",
"laranja"}
for indice, valor in ipairs(frutas) do
print(indice, valor)
end
```

```
-- Exemplo de repeat-until
local num = 1
repeatprint("Número:", num)
    num = num + 1
until num > 3
```

SOBRECARGA DE OPERADORES

Em Lua, a sobrecarga de operadores é implementada através de metatables e metamethods:

Metamethod	Operador	Descrição
__add	+	Adição
__sub	-	Subtração
__mul	*	Multiplicação
__div	/	Divisão
__mod	%	Módulo
__pow	^	Exponenciação
__eq, __lt, __le	"==, <, <=”	Comparação

Isso acontece porque Lua não sabe aplicar operadores em tabelas, então dá a chance ao programador de definir o comportamento via metatables/metamethods, permitindo sobrecarga de operadores.

REFERÊNCIA

- Documentação oficial: http://www.lua.org/docs.html
- Referência: http://www.lua.org/manual/5.1/
- Livro online gratuito (Lua 5.0): http://www.lua.org/pil/contents.html
- Repositório GitHub: https://github.com/aliciamonteiro/Lua

OBRIGADO

pela atenção!

INTEGRANTES

Alicia Monteiro, Eduardo Couto, João Vitor Fernandes,
Kleiton Josivan e Robert Danilo

alicia20230028449@alu.uern.br
eduardo20230028743@alu.uern.br
joao20230021214@alu.uern.br
kleiton20230036226@alu.uern.br
robert20230023200@alu.uern.br

ALGUMA DÚVIDA?