

Linguagens de Programação

Lua 5.1

Prof. Bruno Silvestre
brunoos@inf.ufg.br



Tipos e Valores: table

- É a implementação de arrays associativos

t = {}



TABELA

KEY	VALUE



Tipos e Valores: table

- É a implementação de arrays associativos
 - Exceto *nil*, qualquer valor pode ser índice
 - Pode armazenar qualquer valor, inclusive *nil*

```
t = {}
t[1] = "hello"
t[true] = false
```

TABELA

KEY	VALUE
1	"hello"
true	false



Tipos e Valores: table

- É a implementação de arrays associativos
 - Exceto *nil*, qualquer valor pode ser índice
 - Pode armazenar qualquer valor, inclusive *nil*

```
t = {}
t[1] = "hello"
t[true] = false
t[true] = nil
print( t[print] )

t["f"] = print
t["f"](t)
```

TABELA

KEY	VALUE
1	"hello"
true	nil
"f"	<FUNCTION>



Tipos e Valores: table

- Tabelas são *objetos*
 - Alocadas dinamicamente
 - Crescem com a demanda
 - Variáveis são referências para a tabela

```
t = {}  
a = t  
  
a[1] = 1+2  
print( t[1] )  
  
t = nil  
a = nil
```

5

Tipos e Valores: table

- Podemos ter variáveis ou expressões dentro do colchetes: o resultado é usado como chave

```
t = {}  
a = "T"  
t[a] = 4+1  
print( t["T"] )  
  
t[a..a..a] = 19  
print( t["TTT"] )  
  
i = 10  
t[ i+1 ] = {}
```

6

Tipos e Valores: table

- Chaves inexistentes retornam *nil*, ou seja, não dá erro!

```
t = {}                -- tabela vazia

print( t[1] )         -- nil
print( t["bla"] )     -- nil

a = 10
print( t[a] )         -- nil
```



table: Construção de Tipos

- Arrays
- Matrizes
- Registros
 - Estruturas de dados
 - Lista, fila, pilha, árvore, grafos, etc.
 - Objetos
 - Pessoa, aluno, professor, cliente, venda, etc.



Array e Matrizes



9

Arrays

- 1-based
- Índices no intervalo 1 até N (inteiro)
- Operador de tamanho: #

```
a = {}  
a[1] = 10  
a[2] = 'nada'  
a[3] = true  
  
print(#a)
```



10

Arrays

- 1-based
- Índices no intervalo 1 até N (inteiro)
- Operador de tamanho: #

```
a = {}
a[1] = 10
a[2] = 'nada'
a[3] = true
a[5] = 7
a[6] = false

print(#a)
```

```
a = {}
a['1'] = 10
a['2'] = 'nada'
a['3'] = true

print(#a)
```



CUIDADO!

Arrays

- 1-based
- Índices no intervalo 1 até N (inteiro)
- Operador de tamanho: #

```
a = {}
a[#a+1] = 10
a[#a+1] = 'nada'
a[#a+1] = true
a[#a+1] = 7
a[#a+1] = false

print(#a)
```

Arrays

- Podemos implementar uma pilha usando # ?
 - push, pop, peek ?

```
a = {}
a[#a+1] = 10
a[#a+1] = 'nada'
a[#a+1] = true
a[#a+1] = 7
a[#a+1] = false

print(#a)
```



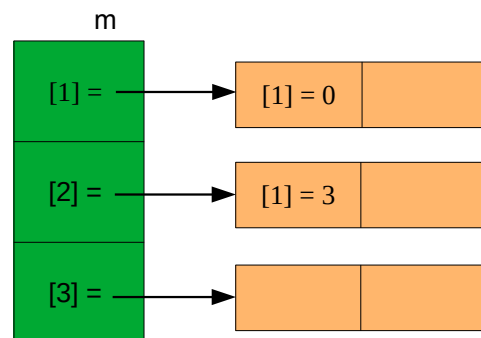
Matrizes

- Matrizes = arrays de arrays (parecido com Java)

```
-- m[3][2]
```

```
m      = {}
m[1]    = {}
m[2]    = {}
m[3]    = {}

m[1][1] = 0
m[2][1] = 3
```



Matrizes

- Matrizes = arrays de arrays (parecido com Java)
- Arrays de 3, 4, 5, ... , N dimensões?

```
-- m[4][3][2]
```

```
m      = {}
```

```
m[1]   = {}
```

```
m[2]   = {}
```

```
m[3]   = {}
```

```
m[4]   = {}
```

```
m[1][1] = {}
```



Registros



Registro

- A tabela é a estrutura, os campos são as chaves


```
aluno = {}
aluno["nome"] = 'Pedro'
aluno["idade"] = 20
aluno["endereço"] = {}
aluno["endereço"]["rua"] = 'Rua 136'
aluno["endereço"]["numero"] = 38
aluno["endereço"]["cep"] = '75382-384'
```



Registro

- A tabela é a estrutura, os campos são as chaves

KEY	VALUE
"nome"	"Pedro"
"idade"	20
"endereço"	



KEY	VALUE
"rua"	"Rua 136"
"numero"	38
"cep"	"75382-384"



Registro

- Lua possui um açúcar sintático:

`t["campo"] ↔ t.campo`

```
pessoa = {}  
pessoa["idade"] = 18  
  
print(pessoa["idade"])  
print(pessoa.idade)  
  
pessoa.idade = 20  
print(pessoa["idade"])
```



Registro

- Podemos simular as *structs* de C

```
aluno = {}  
aluno.nome = 'Pedro'  
aluno.idade = 20  
aluno.endereco = {}  
aluno.endereco.rua = 'Rua 136'  
aluno.endereco.numero = 38  
aluno.endereco.cep = '75382-384'  
  
tmp = aluno.endereco  
tmp['numero'] = 182
```



Construtor de Tabela



24

Construtor de Tabela

```
days = {'SEG', 'TER', 'QUA', 'QUI', 'SEX'}
```

```
days = {}  
days[1] = 'SEG'  
days[2] = 'TER'  
days[3] = 'QUA'  
days[4] = 'QUI'  
days[5] = 'SEX'
```



25

Construtor de Tabela

```
days = {'SEG', 'TER'; 'QUA', 'QUI'; 'SEX', }
```

```
days = {}  
days[1] = 'SEG'  
days[2] = 'TER'  
days[3] = 'QUA'  
days[4] = 'QUI'  
days[5] = 'SEX'
```



Construtor de Tabela

```
person = {'Pedro', 25, true, {}}
```

```
person = {}  
person[1] = ???  
person[2] = ???  
person[3] = ???  
person[4] = ???
```



Construtor de Tabela

- Podemos usar variáveis ou expressões na criação de uma tabela

```
nome = 'Pedro'
idade = 25
socio = true
endereco = {}

-- pack
person = {idade+1, nome, endereco, socio}

print(person[1])
print(person[2])
print(person[3])
print(person[4])
```



Construtor de Tabela

- Podemos informar o **índice** explicitamente na criação das tabelas

```
days = {
  [2] = 'TER',
  [3] = 'QUA',
  [5] = 'SEX',
  [1] = 'SEG',
  [4] = 'QUI',
}
```

```
person = {
  ['nome'] = 'Pedro',
  ['idade'] = 25+1,
  ['socio'] = true,
  ['endereco'] = {
    ['rua'] = 'Rua 123',
    ['cep'] = '37362-392',
  },
}
```



Construtor de Tabela

- Se o índice for uma string (sem espaço), podemos retirar as aspas e os colchetes

```
person = {
  ['nome'] = 'Pedro',
  ['idade'] = 25,
  ['socio'] = true,
  ['endereco'] = {
    ['rua'] = 'Rua 123',
    ['cep'] = '37362-392',
  }
}
```

```
person = {
  nome = 'Pedro',
  idade = 25,
  socio = true,
  endereco = {
    rua = 'Rua 123',
    cep = '37362-392',
  }
}
```

EQUIVALENTES



32

Construtor de Tabela

```
person = {
  nome = 'Pedro',
  idade = 25,
  socio = true,
  endereco = {
    rua = 'Rua 123',
    num = 42,
    cep = '37362-392',
  }
}
```

Equivalentes

```
person = {}
person.nome = 'Pedro'
person.idade = 20
person.endereco = {}
person.endereco.rua = 'Rua 136'
person.endereco.numero = 38
person.endereco.cep = '75382-384'
```



33

Módulo *table*

Funções	Descrição
table.concat	
table.insert	
table.maxn	
table.remove	
table.sort	



Estruturas de Controle



if ... then ... [else ...] end

```
if x > 0 then  
  x = x + 1  
  print(x)  
end
```

```
if a > b then  
  max = a  
else  
  max = b  
end
```

```
if x > 0 and y < 10 or nome == 'admin' then  
  x = x + 1  
  print(x)  
end
```



while e repeat

```
while x < 10 do  
  x = x + 1  
  print(x)  
end
```

```
repeat  
  x = x + 1  
  print(x)  
until x == 10
```



for Numérico

- *inicio*, *fim* e *inc* podem ser valores constantes, expressões ou retorno de função
- O incremento é opcional, o padrão é 1

```
for <var> = <inicio>, <fim> [, <inc>] do  
  
end
```



for Numérico

```
for <var> = <inicio>, <fim> [, <inc>] do  
  
end
```

```
for i = 1, 10 do  
  print(i)  
end
```

```
for i = 1, 10, 2 do  
  print(i)  
end
```

```
for i = 12, 4, -3 do  
  print(i)  
end
```



for Genérico

- *iter* é uma função que cria um iterador
- *var1, ..., varN* são variáveis para os valores retornados pelo iterador

```
for <var1>, ..., <varN> in <iter> do  
  
end
```



for Genérico

- Iteradores para tabela:
 - `pairs(tabela)` → percorre toda a tabela
 - Sem uma ordem definida
 - `ipairs(tabela)` → percorre os índices numéricos do array
 - Começa de 1

```
for k, v in pairs(t) do  
  print(k, v)  
end
```

```
for k, v in ipairs(t) do  
  print(k, v)  
end
```



for Genérico

```
aluno = {}
aluno.nome = 'Pedro'
aluno.idade = 20
aluno.matricula = '201001'

for k, v in pairs(aluno) do
  print(k, v)
end
```

Possível saída do programa

```
→ idade      20
→ nome       Pedro
→ matricula  201001
```



for Genérico

```
a = {}
a[5] = 10
a[4] = 'nada'
a[3] = true
a[2] = 7
a[1] = false

for k, v in ipairs(a) do
  print(k,v)
end
```

Saída do programa

```
→ 1      10
→ 2      nada
→ 3      true
→ 4      7
→ 5      false
```



for Genérico

- pairs → mostra todos (não garante ordem)
- ipairs → itera até o 3

```
a = {}  
a[1] = 10  
a[2] = 'nada'  
a[3] = true  
a[8] = 7  
a[9] = false  
  
for k, v in pairs(a) do print(k,v) end  
for k, v in ipairs(a) do print(k,v) end
```

