tables

(e ponteiros mas mto pouco)

semana 8


JOY FIELDING
JOY FIELDING
SPARTACUS
MAX
CLIVE CUSSLER
FLOOD TIDE
CLIVE CUSSLER
CITY OF GOLD
CHARITY
Beaches
Tom Clancy Debt of Hono

Uma **Table** é uma *estrutura de dados* que utilizaremos para organizar os dados que manipularemos no decorrer da execução de nosso programa.

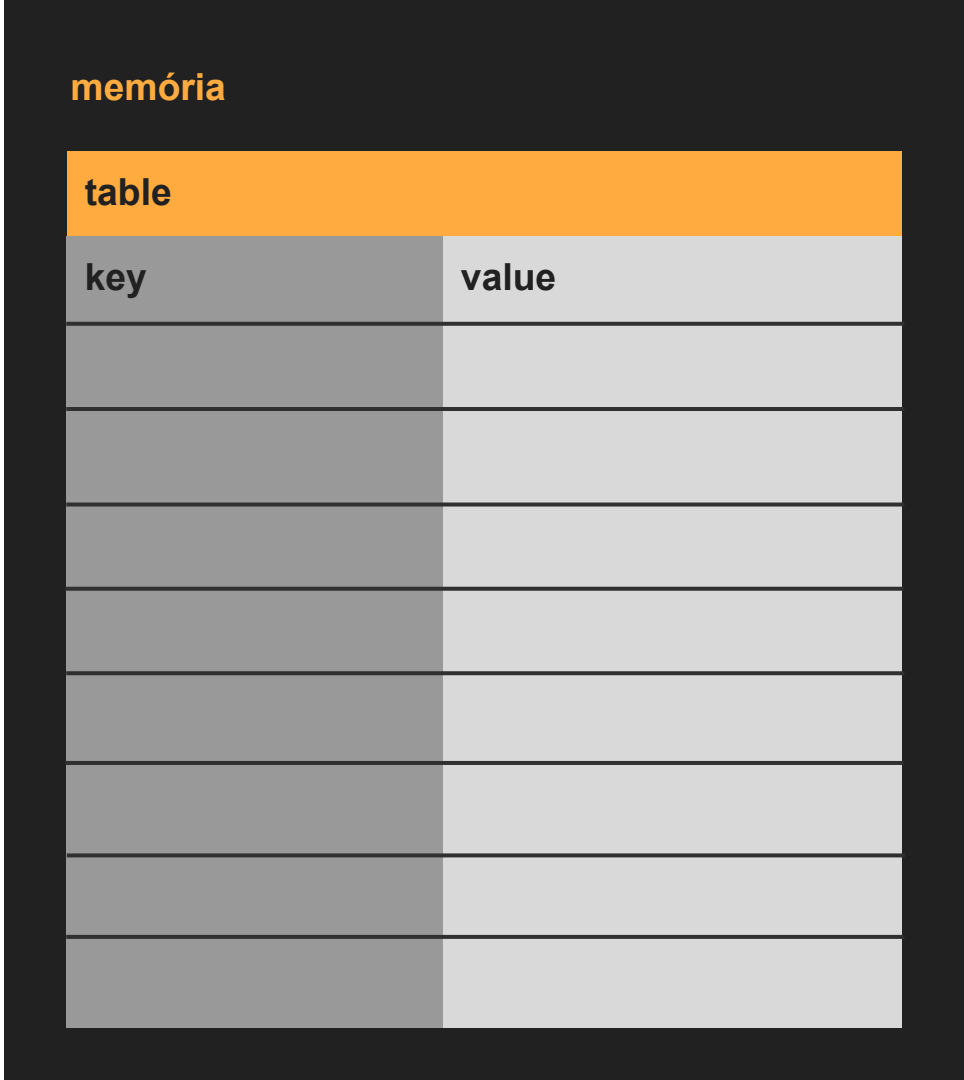
Estrutura de dados permitem que dados sejam armazenados e acessados com mais facilidade na memória RAM.

Elas também nos ajudam a manter nossa programação compacta e compreensível.

```
local table = {}
```

criamos uma **Table**, guardamos uma referência a ela em *table*. Esta **Table** está vazia.

local table = {}



Tables armazenam dados em *Keys* e *Values.*

Nós utilizamos as *Keys* para acessar os *Values*.

```
local nomes = {}
```

```
local nomes = {}
nomes[1] = “João”
```

memória

nomes	
key	**value**
1	“João”

```
local nomes = {}
nomes[1] = “João”
nomes[2] = “Maiara”
```

memória

nomes	
key	**value**
1	“João”
2	“Maiara”

```
local nomes = {}
nomes[1] = “João”
nomes[2] = “Maiara”
nomes[“a”] = “Rocco”
```

memória

nomes	
key	**value**
1	“João”
2	“Maiara”
“a”	“Rocco”

Keys não precisam ser apenas números ou strings, elas podem ser qualquer valor de Lua, exceto *nil*.

```
local x = 10
local nomes = {}
nomes[1] = “João”
nomes[2] = “Maiara”
nomes[x] = “Rocco”
```

memória

nomes	
key	**value**
1	“João”
2	“Maiara”
10	“Rocco”

```
local nomes = {}
nomes[1] = “João”
nomes[2] = “Maiara”
```

memória

nomes	
key	**value**
1	“João”
2	“Maiara”

local nomes = {“João”, “Maiara”}

memória

nomes	
key	**value**
1	“João”
2	“Maiara”

```
local nomes = {}
nomes[1] = “João”
nomes[2] = “Maiara”
nomes[“a”] = “Rocco”
```

memória

nomes	
key	**value**
1	“João”
2	“Maiara”
“a”	“Rocco”

```
local nomes = {
	“João”,
	“Maiara”,
	a = “Rocco”
}
```

memória

nomes	
key	**value**
1	“João”
2	“Maiara”
“a”	“Rocco”

```
local nomes = {
    “João”,
    “Maiara”,
    a = “Rocco”
}
```

memória

nomes	
key	**value**
1	“João”
2	“Maiara”
“a”	“Rocco”

Embora não usamos aspas, a *key ‘a’* é interpretada como uma string (apenas nas expressões construtoras).

```
local nomes = {
    “João”,
    “Maiara”,
    a = “Rocco”,
    b = “Luísa”,
    “Carmen”
}
```

memória

nomes	
key	**value**
1	“João”
2	“Maiara”
“a”	“Rocco”
“b”	“Luísa”
3	“Carmen”

Não podemos declarar *keys* numéricas na *expressão construtora*, nem podemos utilizar strings que começam com números!

```
local nomes = {
        “João”,
        “Maiara”,
        3 = “Rocco”,
        b = “Luísa”,
        “Carmen”
}
```

ERRO

```
local nomes = {
	“João”,
	“Maiara”,
	3abc = “Rocco”,
	b = “Luísa”,
	“Carmen”
}
```

ERRO

```
local nomes = {
	“João”,
	“Maiara”,
	abc3 = “Rocco”,
	b = “Luísa”,
	“Carmen”
}
```

FUNCIONA

para acessar um valor em nossa tabela, usamos a sua key:

table[*key*] ➡ *value*

```
local nomes = {
        “João”,
        “Maiara”,
        a = “Rocco”,
        b = “Luísa”,
        “Carmen”
}

print(nomes[1])
```

console

João

nomes[1] ➡ “João”

```
local nomes = {
	“João”,
	“Maiara”,
	a = “Rocco”,
	b = “Luísa”,
	“Carmen”
}

print(nomes[1])
print(nomes[2])
```

console

```
João
Maiara
```

nomes[2] ➡ “Maiara”

```
local nomes = {
	“João”,
	“Maiara”,
	a = “Rocco”,
	b = “Luísa”,
	“Carmen”
}

print(nomes[1])
print(nomes[2])
print(nomes[“a”])
```

console

João
Maiara
Rocco

nomes[“a”] ➡ “Rocco”

```
local nomes = {
	“João”,
	“Maiara”,
	a = “Rocco”,
	b = “Luísa”,
	“Carmen”
}

print(nomes[1])
print(nomes[2])
print(nomes[“a”])
print(nomes[3])
```

console

```
João
Maiara
Rocco
Carmen
```

nomes[3] ➡ “Carmen”

```
local nomes = {
	“João”,
	“Maiara”,
	a = “Rocco”,
	b = “Luísa”,
	“Carmen”
}

print(nomes[“b”])
```

console

Luísa

```
local nomes = {
        “João”,
        “Maiara”,
        a = “Rocco”,
        b = “Luísa”,
        “Carmen”
}

print(nomes[“b”])
print(nomes.b)
```

console

```
Luísa
Luísa
```

nomes[“b”] == nomes.b

table[x] ~= table.x
table[“x”] == table.x

```
local x = 10
local nomes = {}
nomes[x] = “João”
nomes[“x”] = “Rocco”

print(nomes[10])
print(nomes.x)
```

console

```
João
Rocco
```

nomes[“1”] ~= nomes.1

ERRO

Para acessar um valor usando o ‘.’ a *key* precisa ser uma String que não pode ser iniciada por um dígito!

nomes[“abc1”] == nomes.abc1

```
local nomes = {abc123 = “João”}
nomes[1] = “Kevin”
nomes[“1”] = “Lucca”

print(nomes.abc123)
print(nomes[1])
print(nomes[“1”])
```

console

```
João
Kevin
Lucca
```

Tables podem ser armazenadas
dentro de outras tables!

```
local monstros = {
	orcs = {“Cláudio”, “José”},
	goblins = {“Tomas”},
	dragoes = {“Ernesto”}
}

print(monstros[“orcs”][2])
```

console

José

```
local monstros = {
	orcs = {“Cláudio”, “José”},
	goblins = {“Tomas”},
	dragoes = {“Ernesto”}
}

print(monstros[“orcs”][1])
print(monstros.dragoes[1])
```

console

```
José
Ernesto
```

Podemos iterar com facilidade pelas nossas tables usando um loop for!

```
local orcs = {
	“José”, “Maria”, “Otávio”,
	“Miguel”, “João”
}

for i = 1, 5 do
	print(orcs[i])
end
```

console

```
José
Maria
Otávio
Miguel
João
```

```
local numbers = {}
local x = 101

for i = 1, 100 do
	numbers[i] = x
	x = x + 1
end

print(numbers[100])
```

console

```
200
```

```
local numbers = {}
local x = 101

for i = 1, 100 do
	numbers[i] = x
	x = x + 1
end

print(#numbers)
```

console

```
100
```

#table retorna quantos itens existem em *table*

```
local numbers = {}
local x = 101

for i = 1, 100 do
	numbers[i] = x
	x = x + 1
end

for i = 1, #numbers do
	numbers[i] = 0
end

print(numbers[100])
```

console

```
0
```

Lua possui funções que nos ajudam a realizar o mesmo de maneira ainda mais simples.

```
local orcs = {
	“José”, “Maria”, “Otávio”,
	“Miguel”, “João”
}

for i = 1, 5 do
	print(orcs[i])
end
```

console

```
José
Maria
Otávio
Miguel
João
```

```
local orcs = {
	“José”, “Maria”, “Otávio”,
	“Miguel”, “João”
}

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

console

```
José
Maria
Otávio
Miguel
João
```

```
for k, v in ipairs(table) do
    ...
end
```

```
for key, value in ipairs(table) do
    ...
end
```

```
local orcs = {
	“José”, “Maria”, “Otávio”,
	“Miguel”, “João”
}

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

console

```
José
Maria
Otávio
Miguel
João
```

```
local orcs = {
	“José”, “Maria”, “Otávio”,
	“Miguel”, “João”
}

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

console

```
1	José
2	Maria
3	Otávio
4	Miguel
5	João
```

```
for key, value in ipairs(table) do
    ...
end
```

Key e *Value* não precisam ter necessariamente esses nomes!

A função **ipairs** ignora itens cuja *key* não seja numérica. Além disso, a ordem de iteração será sempre numérica

```
local orcs = {
	“José”, “Maria”, “Otávio”,
	“Miguel”, “João”,
	chefe = “Madruga”
}

for index, nome in ipairs(orcs) do
	print(index, nome)
end
```

console

```
1	José
2	Maria
3	Otávio
4	Miguel
5	João
```

```
for key, value in pairs(table) do
    ...
end
```

A função **pairs** funciona da mesma maneira, mas ela itera por todos os itens da table, até mesmo os que possuem *keys* não numéricas. Enquanto a ordem de iteração pode ser numérica, isso não é garantido!

```
local orcs = {
	“José”, “Maria”, “Otávio”,
	“Miguel”, “João”,
	chefe = “Madruga”
}

for index, nome in pairs(orcs) do
	print(index, nome)
end
```

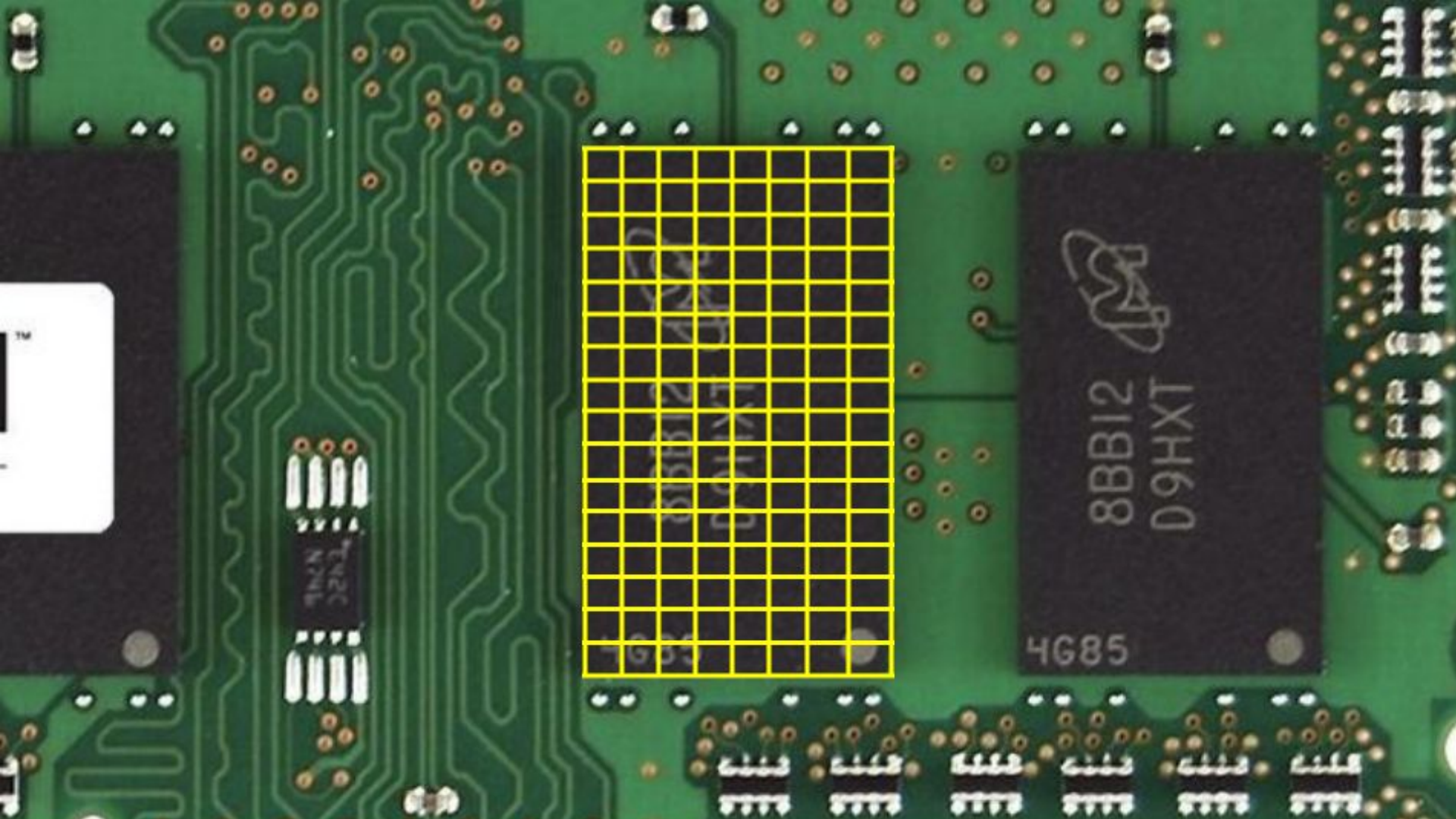
console

```
1	José
3	Otávio
“chefe”	Madruga
2	Maria
5	João
4	Miguel
```

E agora, meus camaradas,
ponteiros…
(o basiquinho pelo menos)


crucial
by Micron
47J6
4G85
8BBI2
D9HXT
4G85
8BBI2
D9HXT
4G85

A memória RAM do nosso computador possui milhares e milhares de bytes. Podemos pensar nos bytes como um grid.


8BBI2
D6HXT
4G85

Os dados que utilizamos são guardados nas casas desse grid. Cada casa possui um *endereço de memória* único, que é um número qualquer
1, 2, 3, 4 …

Através desse endereço de memória, nosso processador sabe onde encontrar cada dado que estamos utilizando em nossos programas.

Por serem muito grandes, endereços de memória são representados por números hexadecimais.

… 00fa84c0 1325ed69 0908fa8b …

Mas por que isso importa?

```
local x = 10
local y = x

print(x)
print(y)
```

console

```
local x = 10
local y = x

print(x)
print(y)
```

console

```
10
10
```

```
local x = 10
local y = x

print(x)
print(y)

x = 0

print(x)
print(y)
```

console

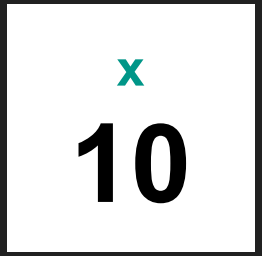
```
10
10
```

```
local x = 10
local y = x

print(x)
print(y)

x = 0

print(x)
print(y)
```

console

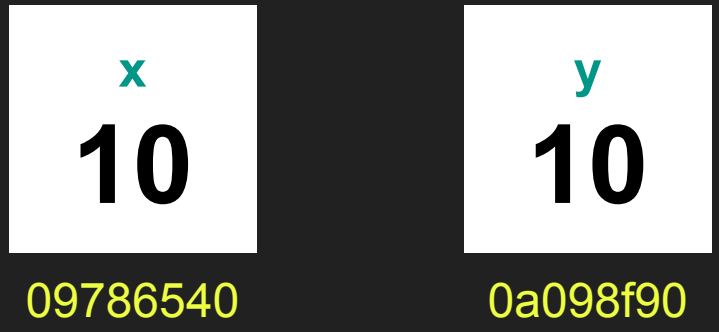
```
10
10
0
10
```

local x = 10

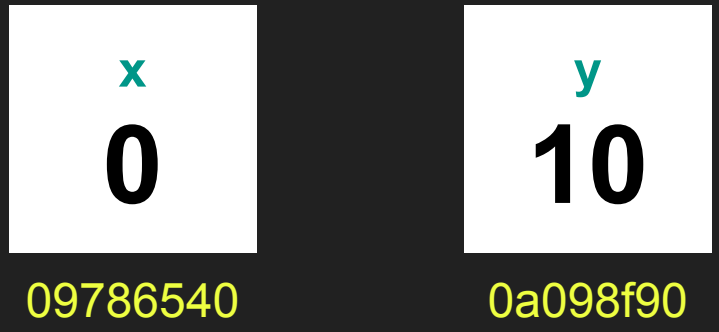


09786540

```
local x = 10
local y = x
```



```
local x = 10
local y = x

x = 0
```



```
local a = {}

print(a)
```

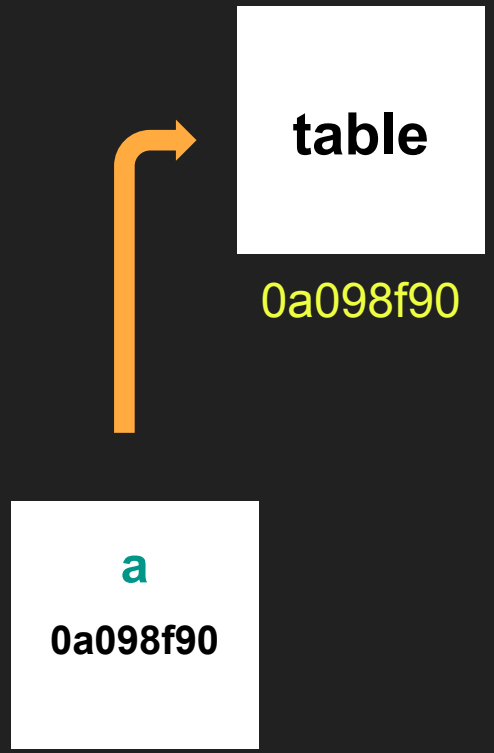
console

```
local a = {}

print(a)
```

console

```
table: 0a098f90
```

local a = {}



A função print imprimiu o endereço de memória no qual reside a nossa table. Este valor muda arbitrariamente a cada execução do nosso programa, pois os endereços de memória são controlados pelo sistema operacional.

```
local a = {}
local b = a

print(a)
print(b)
```

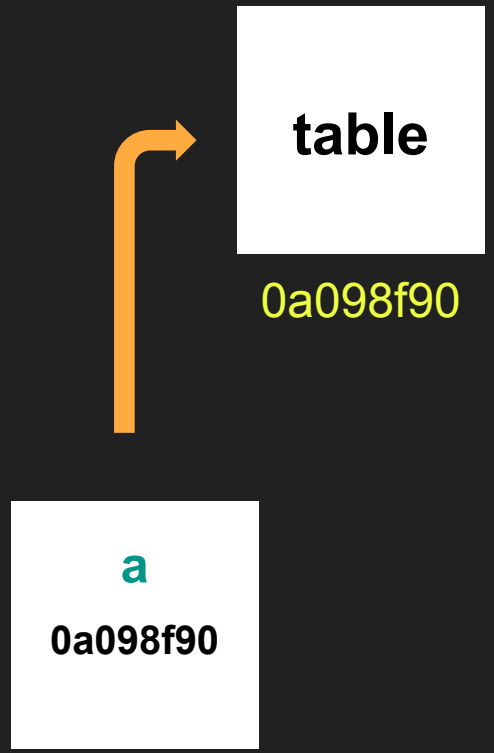
console

```
local a = {}
local b = a

print(a)
print(b)
```

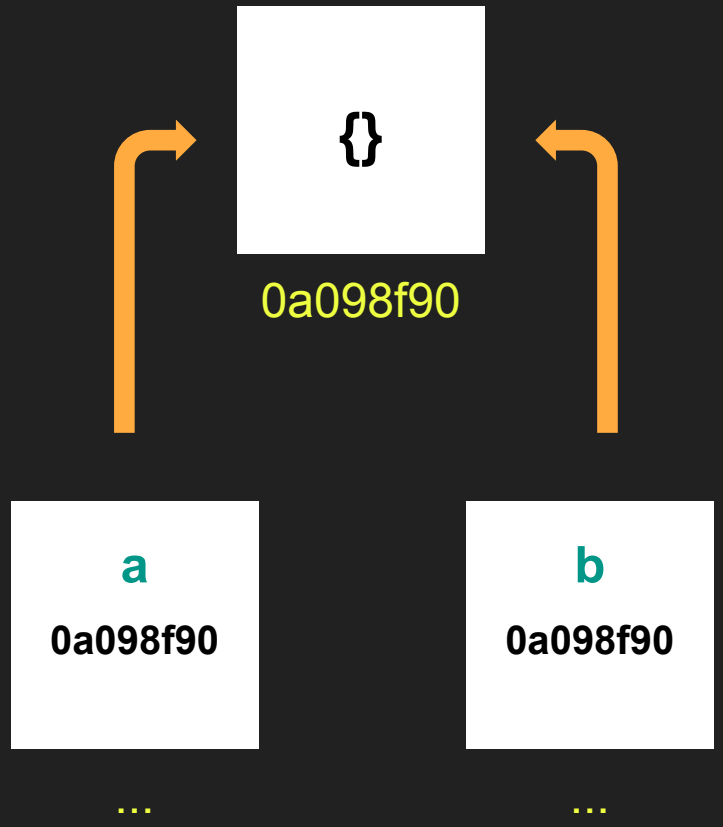
console

```
table: 0a098f90
table: 0a098f90
```

local a = {}



```
local a = {}
local b = a
```



‘a’ e ‘b’ são referências a um único objeto **Table** que reside no endereço de memória **0a098f90**.

```
local a = {}
local b = a

a[1] = “R”

print(a[1])
print(b[1])
```

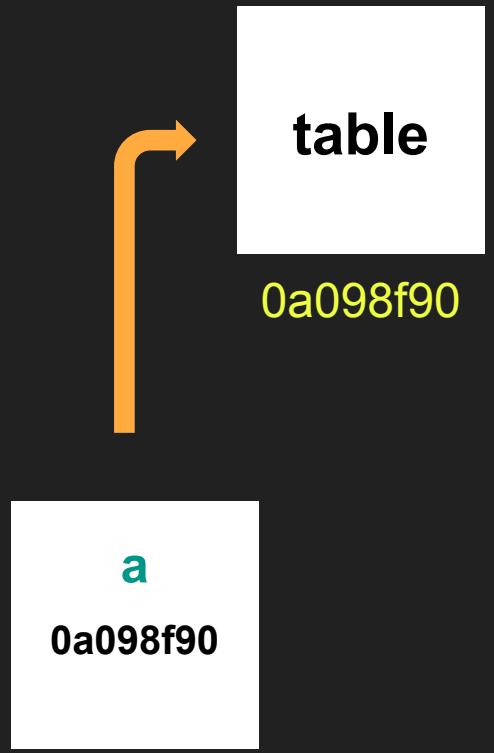
console

```
local a = {}
local b = a

a[1] = “R”

print(a[1])
print(b[1])
```

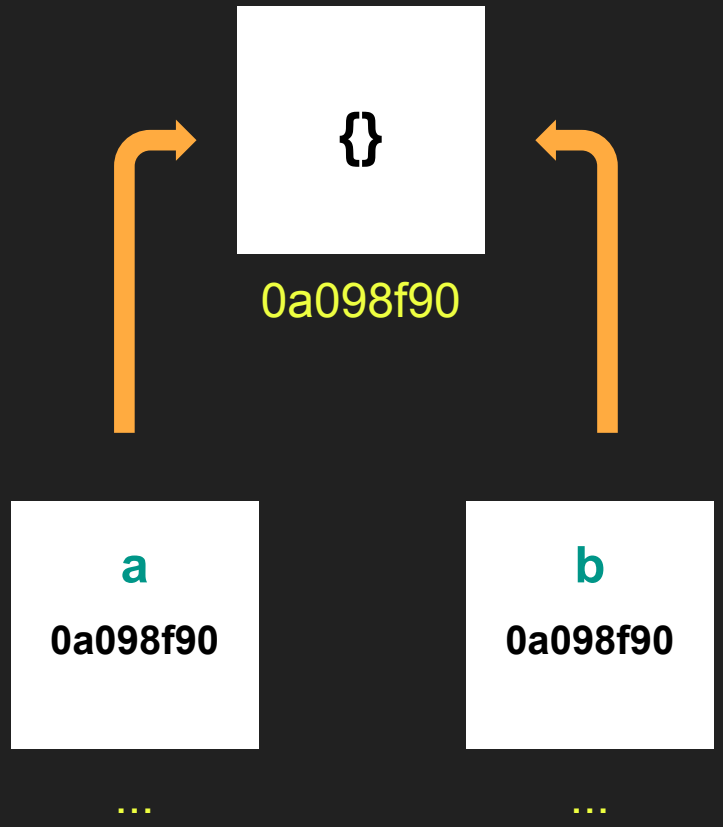
console

```
R
R
```

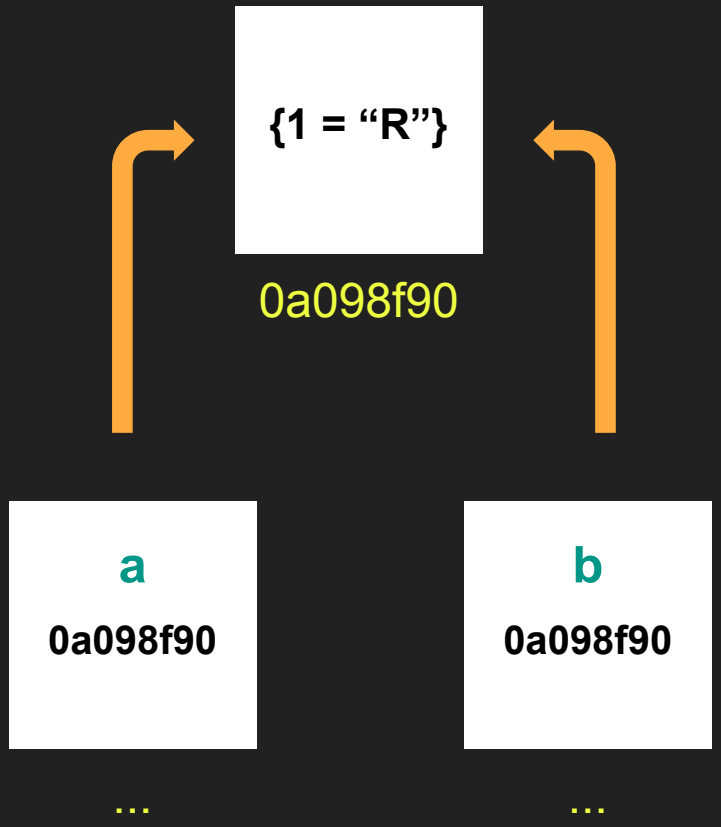
local a = {}



```
local a = {}
local b = a
```



```
local a = {}
local b = a

a[1] = “R”
```



```
local a = {}
local b = a

a[1] = “R”

print(a[1])
print(b[1])
```

console

```
R
R
```

‘a’ e ‘b’ são ponteiros, e apontam para o mesmo endereço, ou seja, para o mesmo objeto na memória. Quando alteramos ‘a’, estamos alterando ‘b’ também.

Próxima aula

04/11 - LÖVE <3