lua

semana 4

```
print("Hello, World!")
```

A execução do programa se inicia no topo do documento, e sua interpretação é feita linha por linha


```
print(                    )
```

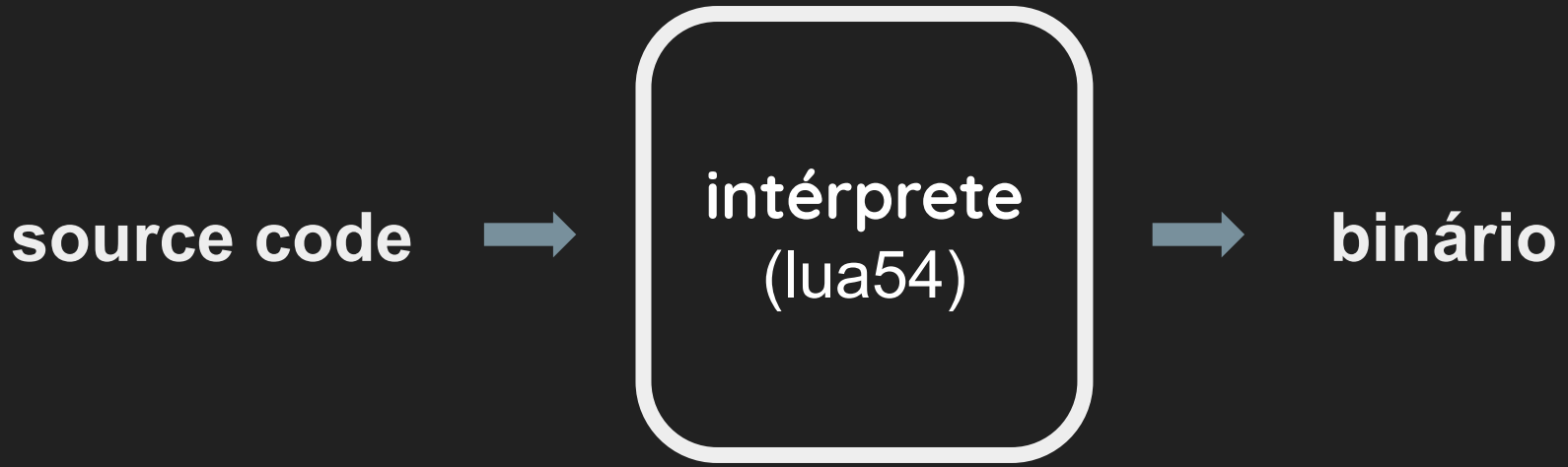
print(Hello, World!)

print(“Hello, World!”)

```
print("Hello, World!")
```

```
print(‘Hello, World!’)
```

Toda sequência de caracteres (*Strings*) deve ser delimitada por aspas simples ou duplas.


source code
intérprete
(lua54)
binário


ask
What's your name?
and wait
say
join
hello,
answer

print(“what’s your name?”)


```
print(“what’s your name?”)
io.read()
```


```
print(“what’s your name?”)
local name = io.read()
```


```
print(“what’s your name?”)
local name = io.read()
print(“Hello, ” .. name)
```

name é uma *String*. Para concatenar strings utilizamos o operador

`..`

```
print("Hello, " .. name)
```


set
counter
to
0

local counter = 0


change
counter
by
1

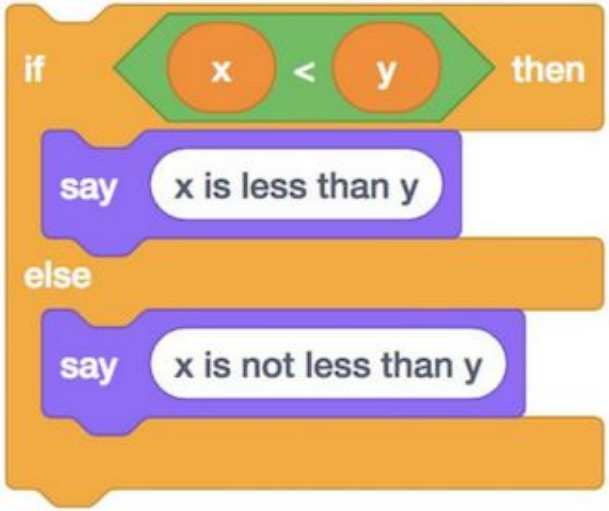
counter = counter + 1

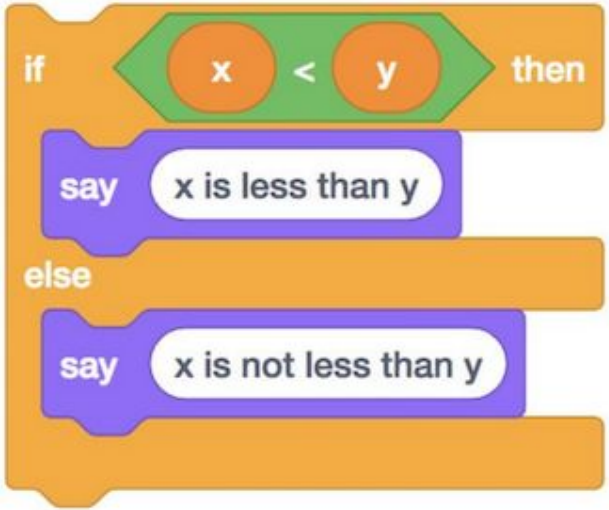
O operador `=` não é um operador de igualdade como em uma equação. Chamado de *assignment operator*, ele significa “copie o valor à direita e guarde-o na variável à esquerda”

Não é necessário estipular *local* nas vezes subsequentes que modificarmos a variável **counter**, pois podemos presumir que já deixamos explícito que ela é uma variável local em sua declaração

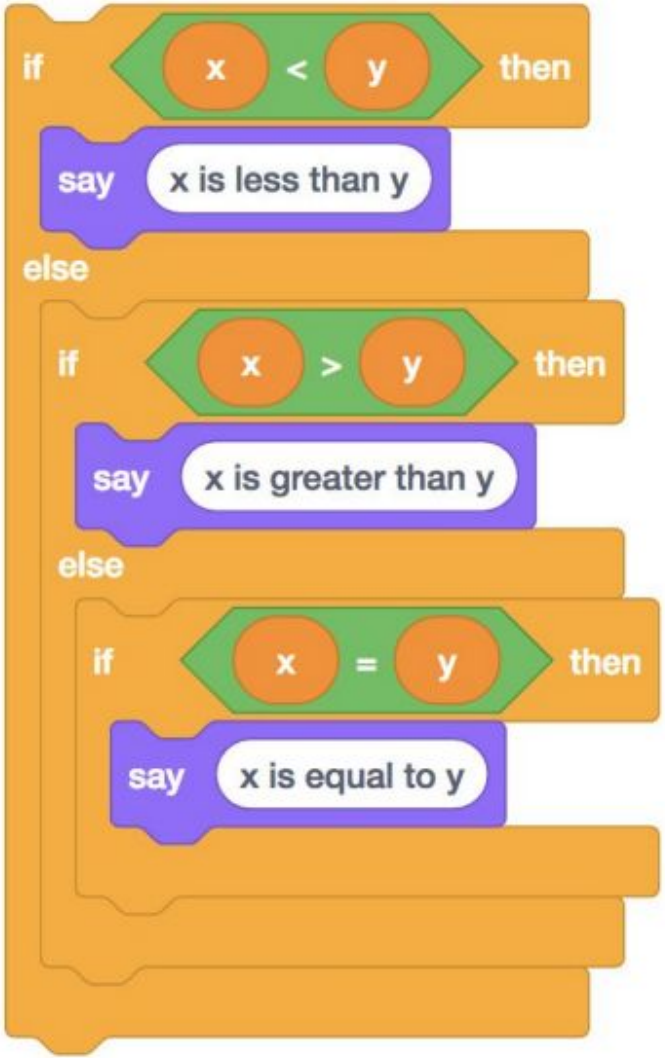

if
x
<
y
then
say
x is less than y

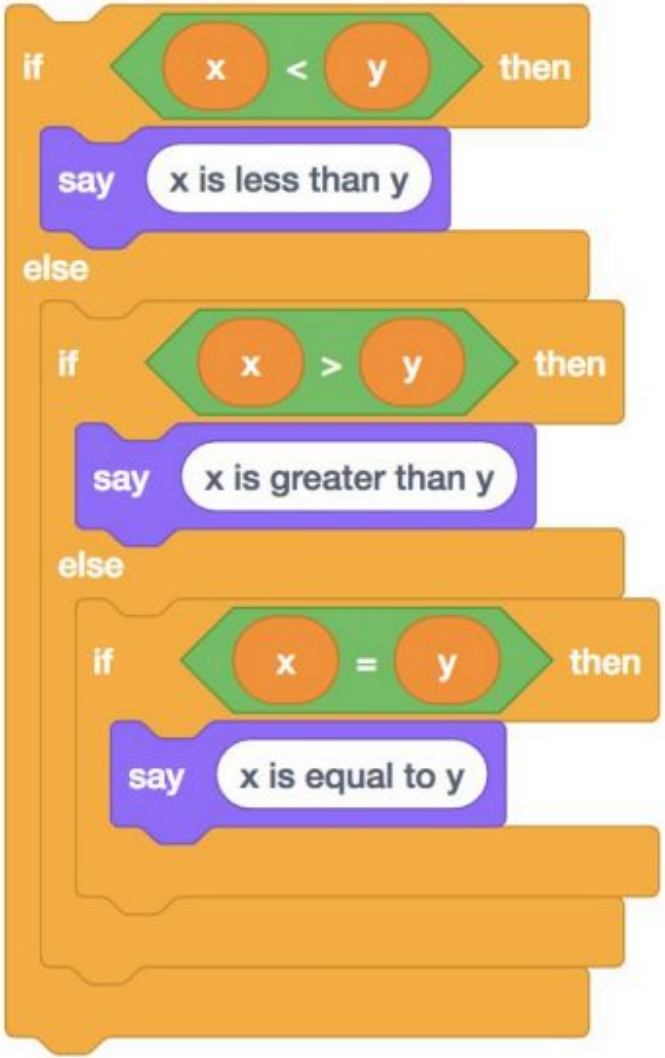

```
if x < y then
      print(“x is less than y”)
end
```


if
x
<
y
then
say
x is less than y
else
say
x is not less than y

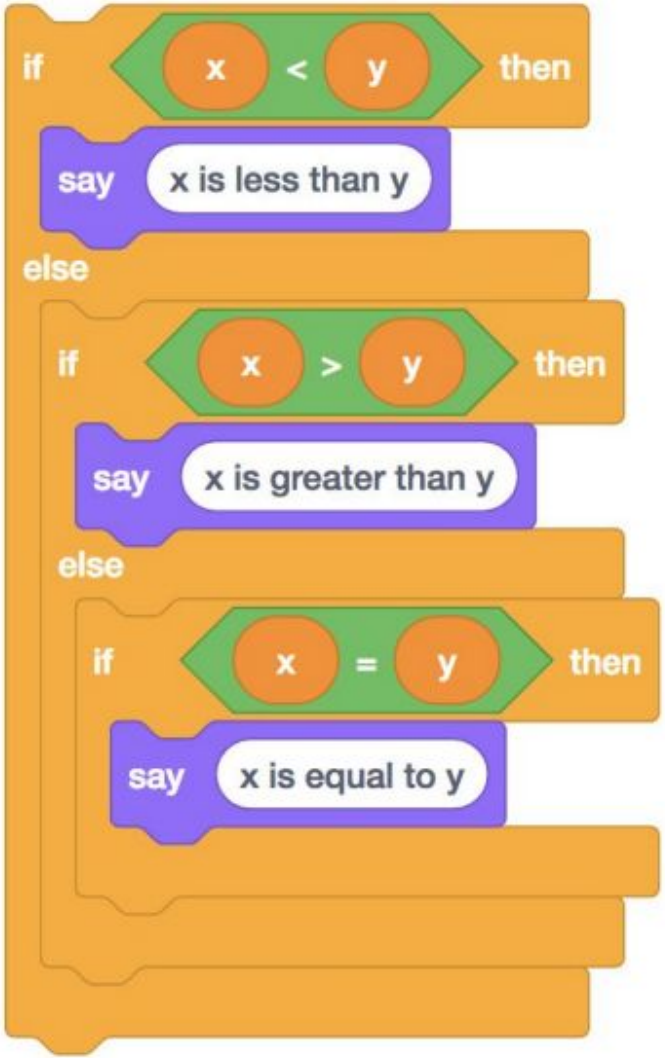



```
if x < y then
      print(“x is less than y”)
else
      print(“x is not less than y”)
end
```


if x < y then
say x is less than y
else
if x > y then
say x is greater than y
else
if x = y then
say x is equal to y



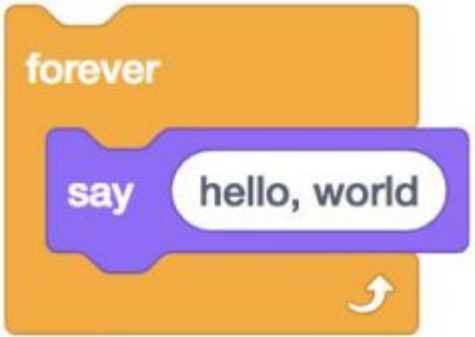

```
if x < y then
      print(“x is less than y”)
elseif x > y then
      print(“x is greater than y”)
elseif x == y then
      print(“x is equal to y”)
end
```




```
if x < y then
      print(“x is less than y”)
elseif x > y then
      print(“x is greater than y”)
else
      print(“x is equal to y”)
end
```

Utilizamos o operador `==` para realizar comparações


forever
say
hello, world




```
while true do
    print(“hello, world”)
end
```

O loop *while* também requer uma condição. Quando essa condição for **falsa** o loop se encerrará. Podemos utilizar a keyword *true* no lugar de uma expressão booleana, assim o loop ocorrerá para sempre.


```
while true do
    print(“hello, world”)
end
```


repeat
50
say
hello, world


```
local i = 0
while i < 50 do
      print(“hello, world”)
      i = i + 1
end
```


```
local i = 50
while i > 0 do
      print(“hello, world”)
      i = i - 1
end
```


```
for i = 0, 49, 1 do
    print(“hello, world”)
end
```

```
for inicial, final, incremento do
    ...
end
```


```
for i = 0, 49 do
    print(“hello, world”)
end
```


```
for i = 50, 1, -1 do
    print(“hello, world”)
end
```

Existem diversos tipos de dados em Lua. Tipos de dados diferentes se comportam de maneiras diferentes.

Nil

Nil é um tipo de dado com um único valor, *nil*. Todas as variáveis que ainda não foram inicializadas equivalem a *nil*. Podemos definir uma variável como *nil* quando queremos deletá-la da memória. Nil é a ausência de um valor útil.

Booleans

O tipo boolean possui dois valores: *true* e *false*. Podem ser usados em expressõe booleanas.

Em lua demais dados também podem ser usados como expressões booleanas. Nesses casos, *nil* e *false* são considerados *false* enquanto todos os demais dados são considerados *true*.

Numbers

Este tipo de dado representa números inteiros ou números reais

4 0.4 4.57e-3 0.3e12 5e+20

Strings

Sequência de caracteres de qualquer tipo.

a = “uma linha”
b = ‘outra linha’
c = “28342938iufhsdkfh”

Tables

Estruturas de armazenamento de dados.

Userdata
Functions
Threads

Escopo de uma variável

Em lua, toda variável é global a não ser que especifiquemos o contrário usando a keyword *local*.

É considerado uma boa prática o uso de variáveis locais. Use variáveis globais apenas quando necessário.

operadores matemáticos

+	adição
-	subtração
/	divisão
*	multiplicação
^	exponenciação
%	módulo

operadores relacionais

== igual
~= diferente
< menor que
> maior que
<= menor ou igual
>= maior ou igual

operadores lógicos

and
or
not

And

Retorna o primeiro argumento se este for falso, se não, retorna o segundo argumento.

Or

Retorna o primeiro argumento se este não for falso, se não, retorna o segundo argumento

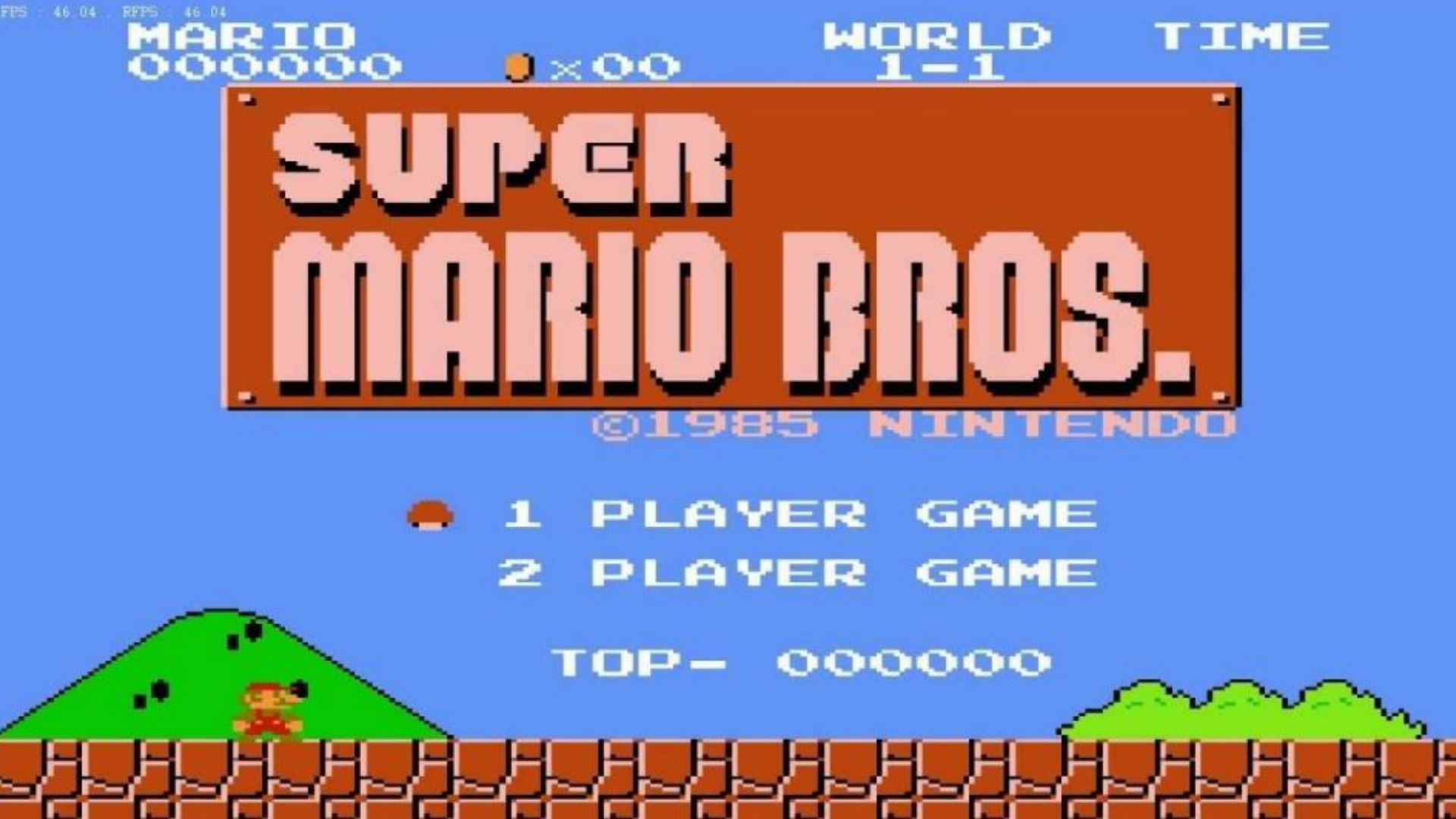
Not

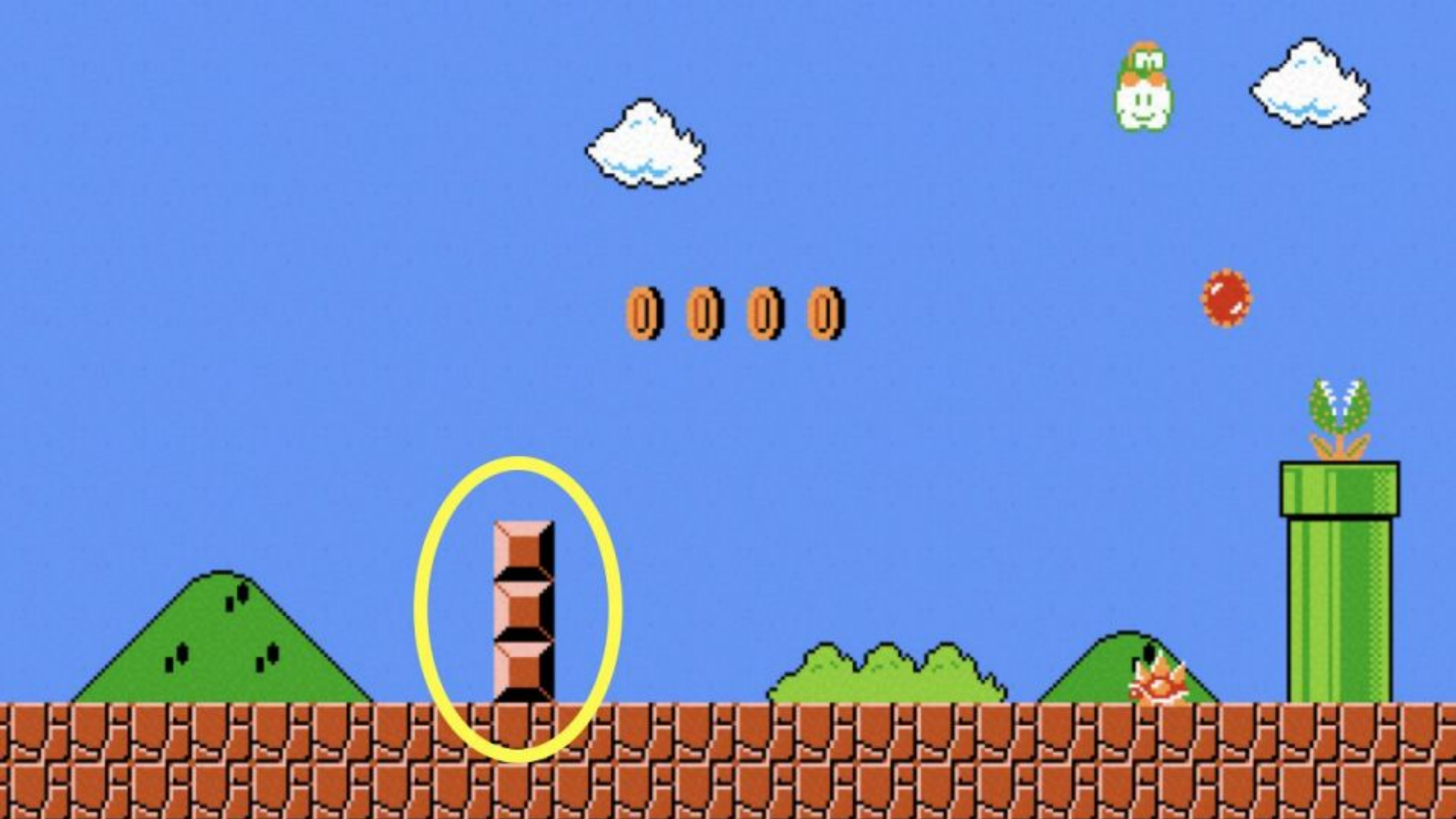
Retorna sempre *true* ou *false*.

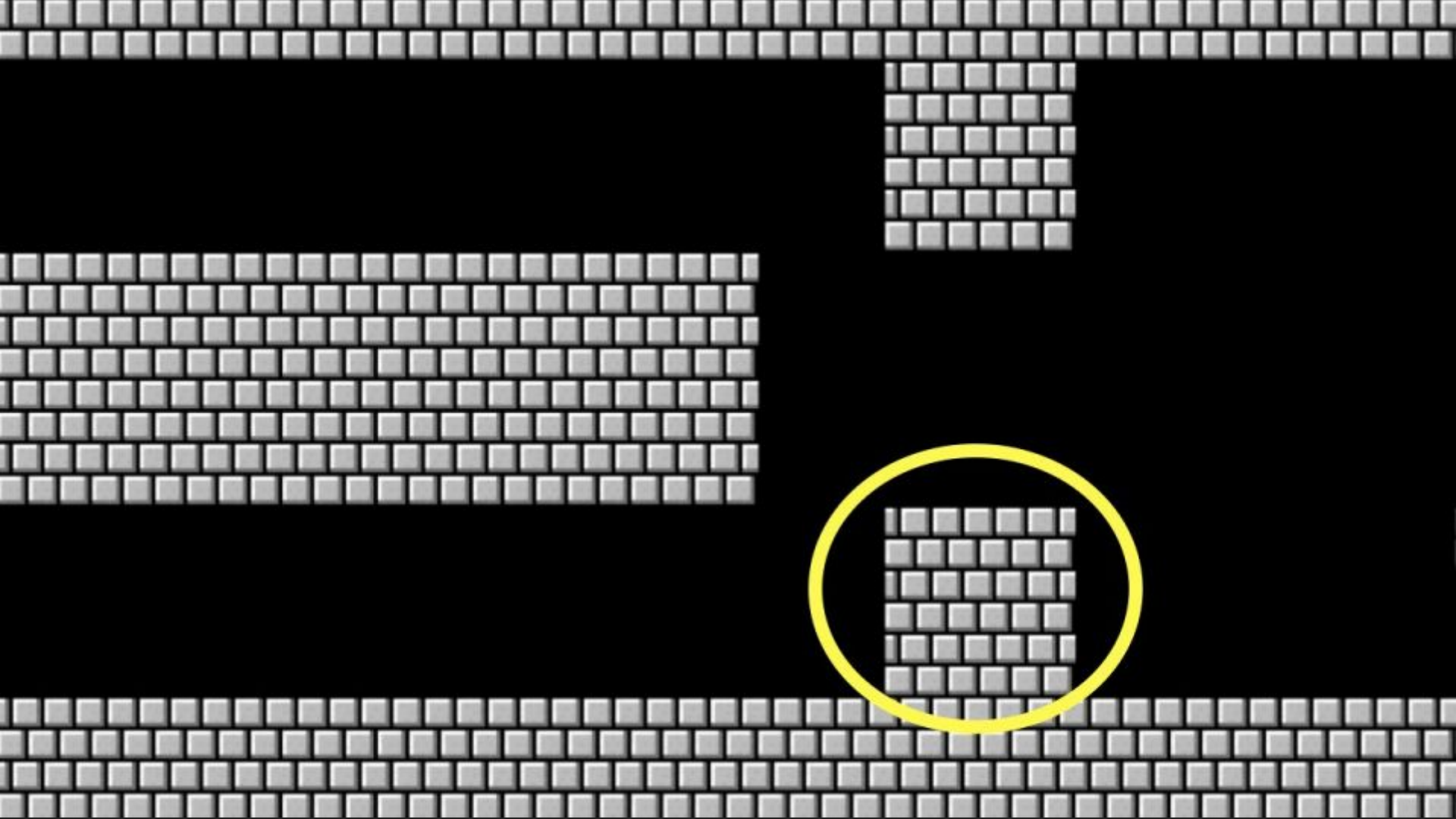
not true = false
not false = true
not nil = true
not 0 = false
not not nil = false

Alguns programas como exemplo

Funções


FPS : 46.04 / REPS : 46.04
MARIO
000000
×00
WORLD
1-1
TIME
SUPER
MARIO BROS.
©1985 NINTENDO
1 PLAYER GAME
2 PLAYER GAME
TOP- 000000





Nosso computador tem uma quantidade finita de memória (RAM). Diferentes tipos de dados possuem diferentes quantidades de bits que armazenam as suas possíveis informações.

integer overflow

floating point imprecision

Atividade proposta:

Desafios - Mario Less ou Mario More

14/10 - Atendimento da atividade (participação opcional)

21/10 - Discussão da atividade e continuação aula