

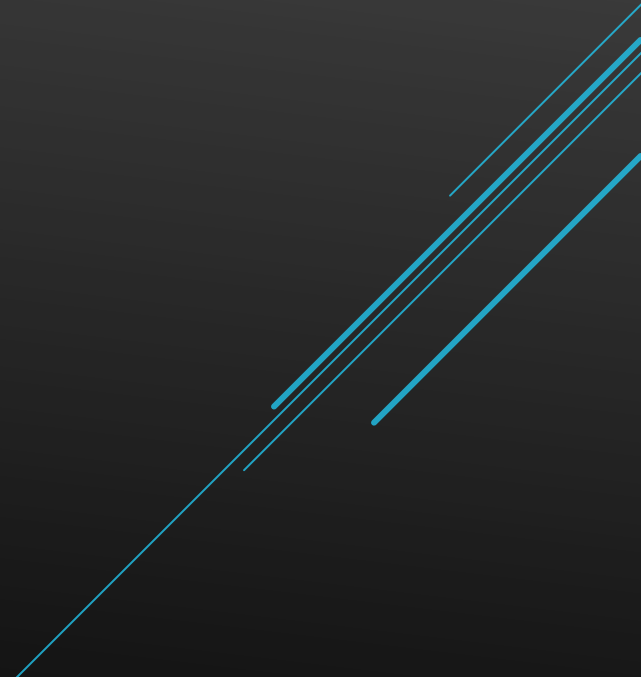


LES BASES DU LUA

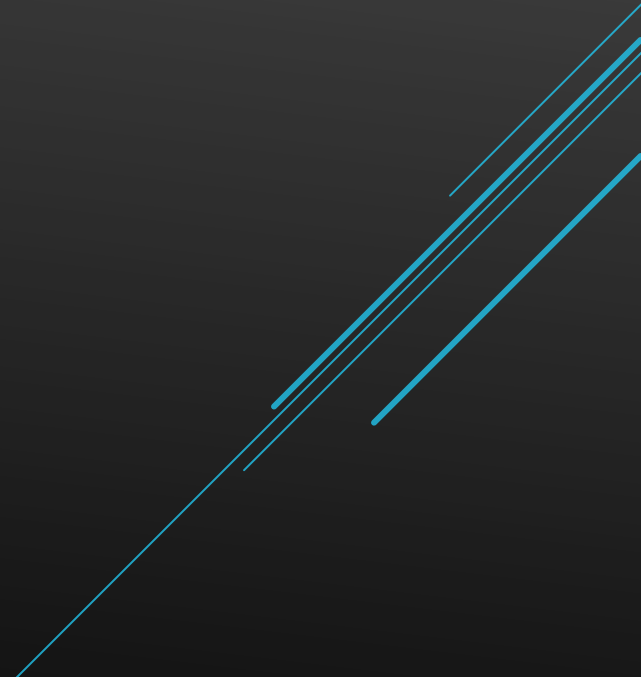


HISTOIRE

- ▶ Première version créé en 1993, officiellement sorti en 1994
- ▶ Nom provient de son prédécesseur SOL (Simple Object Language)



POURQUOI LE LUA ?

- ▶ Langage haut niveau, très simple d'utilisation
 - ▶ Typage dynamique
 - ▶ Cross-platform
 - ▶ Orienté objet
 - ▶ Language compilé, accompagné d'un interpréteur
 - ▶ **Language embarqué**
- 
- Several parallel teal lines of varying lengths and orientations are positioned in the bottom right corner of the slide, creating a modern, abstract graphic element.

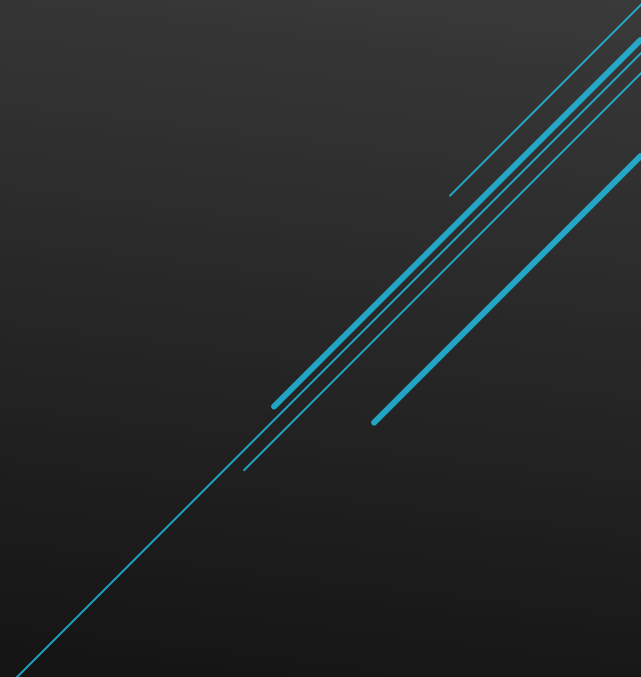
TOUT EST VARIABLE

```
a.lua > ...  
1  #! /usr/bin/env lua  
2  
3  a = function ()  
4      print("a")  
5  end  
6  print(type(a))  
7  a = 1  
8  print(type(a))  
9  
10 print(type(print))  
11 myPrint = print  
12 print = nil  
13 myPrint(type(print))
```

```
pival@DESKTOP-V5FBDBB:~/git/Workshop$ ./a.lua  
function  
number  
function  
nil
```

VARIABLE

- ▶ Nil
 - ▶ Boolean
 - ▶ Number
 - ▶ String
 - ▶ Table
 - ▶ Function

 - ▶ Userdata
 - ▶ Thread
- 
- A series of parallel teal lines of varying lengths and orientations, located in the bottom right corner of the slide, creating a modern, abstract graphic element.

STRING

- ▶ Fonctionne avec les quotes simple (') et double (")
- ▶ Multi line string au format `[[]]`, potentiellement avec des =

```
local a = 'Toto'
print(a)
a = "Tata"
print(a)
a = [[Une string
sur plusieurs lignes]]
print(a)
a = [=[Une string qui contient
des crochets ]] =P]=]
print(a)
```

```
Toto
Tata
Une string
sur plusieurs lignes
Une string qui contient
des crochets ]] =P
```

TABLE

```
a.lua > ...
1  #! /usr/bin/env lua
2
3  local a = { 1, 2, 3, name = "A name", [8] = "number", ["8"] = "string" }
4
5  local b = a
6  b['name'] = "Another name"
7
8  print(a[8])
9  print(a["8"])
10
11 print(a['name'])
12 print(a.name)
13
14 for key, value in pairs(a) do
15     print(key, value)
16 end
```

number	
string	
Another	name
Another	name
1	1
2	2
3	3
8	number
8	string
name	Another name

- Tables sont associatives.
- Le premier élément est à l'index **1**.
- Tables sont copiées par référence
- Il est possible d'accéder aux variables qui ont pour clé une string à l'aide d'un point

FUNCTION

- ▶ Fonctions peuvent être anonymes
- ▶ Fonctions peuvent renvoyer plusieurs valeurs
- ▶ Fonctions peuvent être variadic
- ▶ Le nombre d'argument n'entraîne aucune erreur

```
local function test (funct, ...)  
    return funct(...)  
end  
  
print(test(print, 1, 2, 3))  
print(test(function (a) return a end, 3, 4))  
print(test(function (a, b, c) return c, b, a end, 9))  
print(test(function (...) return table.pack(...)[2] end, 1, 2, 3))
```

1	2	3
3		
nil	nil	9
2		

FUNCTION

- ▶ Parenthèses sont parfois inutiles
- ▶ Valeur par défaut
- ▶ Argument nommé

```
local printTable = function (table, first, last)
    first = first == nil and 1 or first
    last = last or #table

    for i = first, last do
        io.write(tostring(table[i]) .. " ")
    end
    print()
end
```

```
printTable({1, 2, 3, 4, 5, 6, 7, 8})
printTable({1, 2, 3, 4, 5, 6, 7, 8}, 6)
printTable({1, 2, 3, 4, 5, 6, 7, 8}, 1, 4)
```

```
local printTable = function (args)
    local tbl = args.table or args[1] or {}
    local first = args.begin or 1
    local last = args["end"] or #tbl

    for i = first, last do
        io.write(tostring(tbl[i]) .. " ")
    end
    print()
end
```

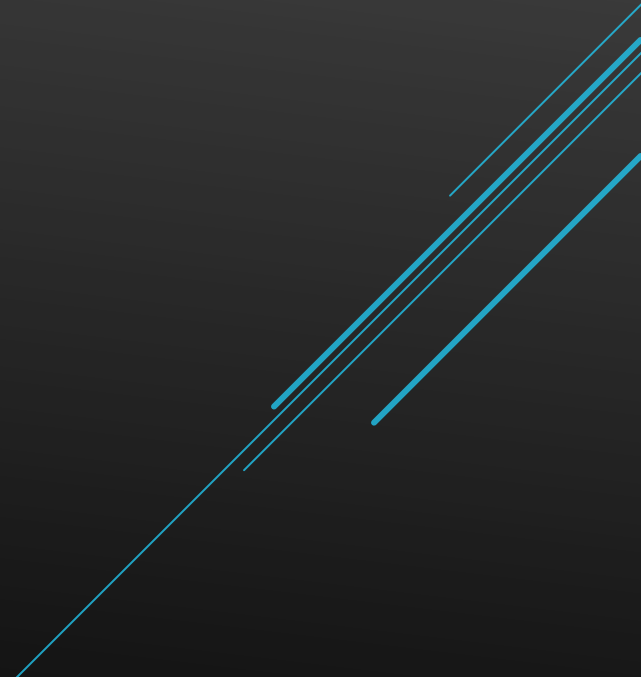
```
printTable{ table = {1, 2, 3, 4, 5, 6, 7, 8} }
printTable{ table = {1, 2, 3, 4, 5, 6, 7, 8}, begin = 6 }
printTable{ {1, 2, 3, 4, 5, 6, 7, 8}, begin = 1, ['end'] = 4 }
```

```
1 2 3 4 5 6 7 8
6 7 8
1 2 3 4
```

OPERATORS

- ▶ Commentaire: `--`
- ▶ Commentaire multi ligne: `-- [[...]]`
- ▶ Operateurs arithmétiques: `+`, `-`, `*`, `/`, `//`, `%`, `^`
- ▶ Operateurs binaire: `&`, `|`, `~`, `<<`, `>>`
- ▶ Operateur de comparaison: `==`, `~=`, `<`, `>`, `<=`, `>=`
- ▶ Operateurs logique: `and`, `or`, `not`
- ▶ Operateur de concatenation: `..`
- ▶ Operateur de longueur: `#`
- ▶ Séparateur de commande: `;`

OPERATORS

- ▶ or
 - ▶ and
 - ▶ < > <= >= ~= ==
 - ▶ |
 - ▶ ~
 - ▶ &
 - ▶ << >>
 - ▶ .
 - ▶ + -
 - ▶ * / // %
 - ▶ unary operators (not # - ~)
 - ▶ ^
- 
- A series of parallel cyan lines of varying lengths and orientations, located in the bottom right corner of the slide, creating a modern, abstract design element.

CONDITION

```
if a < 0 then  
elseif a > 0 then  
else  
end
```

```
var = a > 0 and a or a < 0 and -a or 84
```

Several parallel cyan lines of varying lengths and slopes are positioned in the bottom right corner of the slide, extending from the right edge towards the center.

LOOP

```
while a do  
end
```

```
repeat  
until i > 10
```

```
while i <= 10 do  
end
```

```
for i = 1, 10, 1 do  
end
```

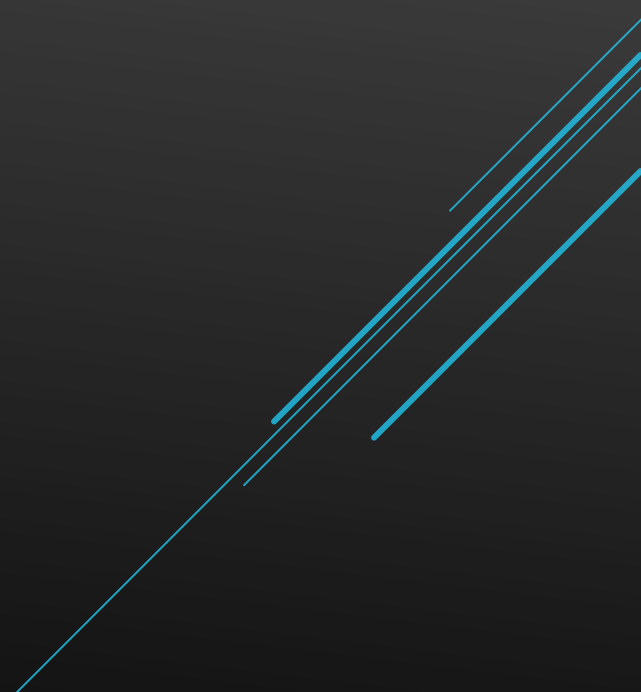
```
local i = 1  
while i <= 10 do  
    i = i + 1  
end
```

```
for index, value in ipairs(table) do  
end
```

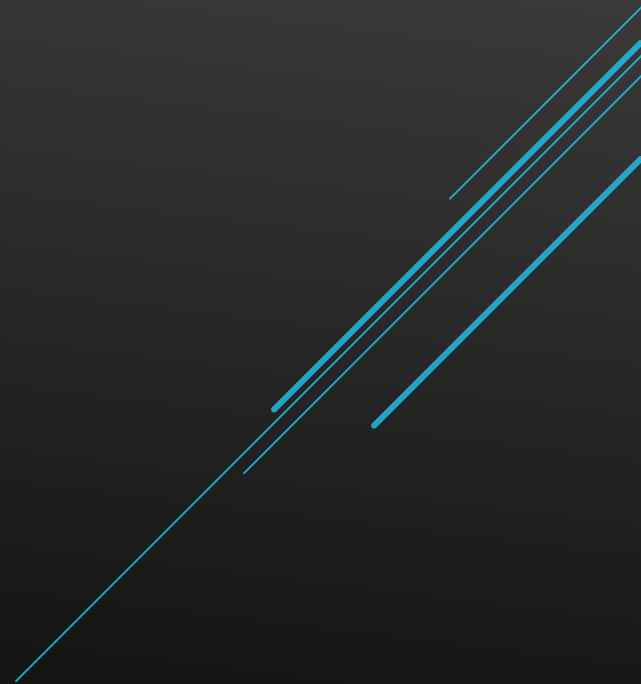
```
local index = 1  
while index <= #table do  
    local value = table[index]  
end
```

BUILT-IN & LIBRAIRIES

- ▶ `Tostring, tonumber, type`
 - ▶ `Print, error, assert`
 - ▶ `Pairs, ipairs`

 - ▶ `string, table, io, math, os, utf8`
 - ▶ `Io.stdin, io.stderr, io.stdout`
 - ▶ `Io.open(file), file:read(), file:write()`
 - ▶ `Table.insert(), table.remove(), table.concat()`
 - ▶ `Pcall(), os.execute(), io.popen(), dofile()`
- 
- Several parallel teal lines of varying lengths and orientations are positioned on the right side of the slide, creating a modern, abstract graphic element.

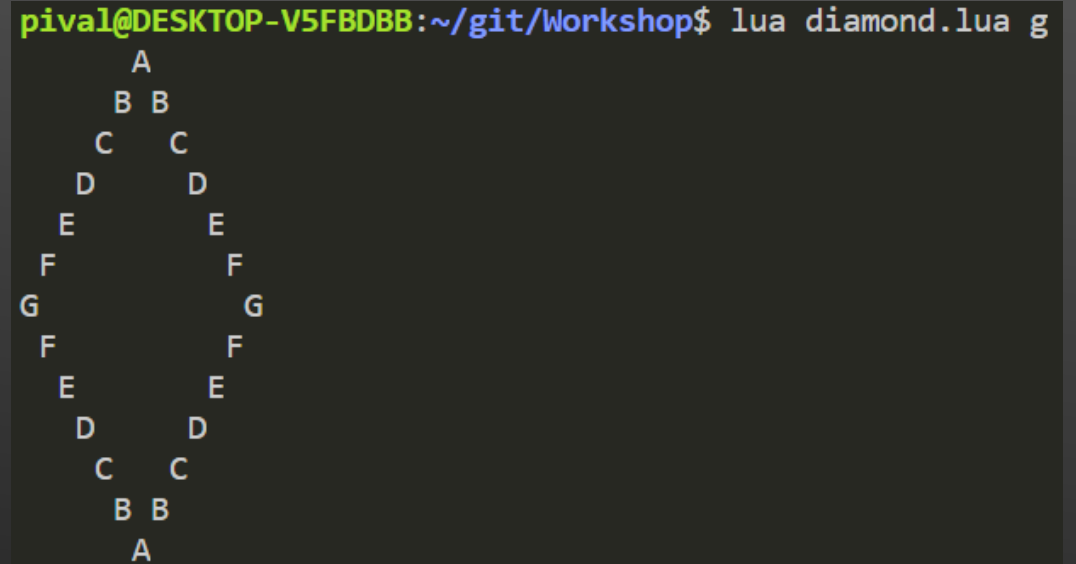
- ▶ <https://www.lua.org/manual/5.4/contents.html>
- ▶ <https://www.lua.org/pil/contents.html>



EXERCISES

- Créez un diamand en utilisant des lettres

```
pival@DESKTOP-V5FBDBB:~/git/Workshop$ lua diamond.lua g
```



- Complétez une map de démineur en ajoutant les chiffres

```
pival@DESKTOP-V5FBDBB:~/git/Workshop$ lua minesweeper.lua
```

