Apprendre le Lua : Session #1

iThorgrim

Apprendre le Lua : Session #1

Qu'est-ce que le Lua ? Une variable ? Une fonction, des arguments ?

Le principe de ses Session est de pouvoir apprendre le Lua directement depuis Open-Wow.eu.

Au cours de cette Session nous verrons donc les bases des sujets suivants :

- Qu'est ce que le Lua ?
- Qu'est ce qu'une variable ?
- Que sont les fonctions et les arguments ?

Qu'est ce que le Lua ?

Le Lua est un langage de développement qui est relativement facile à apprendre.
Si vous avez une certaine connaissance de l'anglais, le Lua n'a déjà presque plus de secrets pour vous.

C'est l'un des rares langages où une "simple traduction" d'une idée en Anglais vous permet d'avoir *(plus ou moins)* une idée de votre code.

Alors ce n'est pas infaillible non plus, je pourrais vous donner de nombreux exemples qui prouvent le contraire, mais cela reste précis dans 80% du code que vous imaginerez.
Le Lua est aussi un langage très "souple", il peut être utilisé partout.

Pour créer des jeux vidéo, des applications de bureau, des IDE et même des sites web.

C'est un langage polyvalent. L'une des forces du Lua est qu'il fait partie de deux familles :

- Les langages lus ou interprétés.
- Les langages non typés.

Les langages lus ou interprétés

Pour rester simple *(sans trop entrer dans les détails)*, vous pouvez exécuter votre code à partir de sa source, sans par le "rendu", la compilation.

Attention, ce n'est pas parce qu'il s'agit d'un langage interprété que vous ne pouvez pas le compiler, cela signifie simplement que vous n'avez pas à passer par la case compilation.

Les langages non typés

Le "typage", pour certains langages, est important. Il s'agit ni plus ni moins d'indiquer quel type de données seras attribué à une variable.

Par exemple, pour une variable appelée "âge", il faut indiquer qu'il d'un *int* (un entier) qui lui seras attribué.

Pas pour le Lua, pour le Lua "âge" peut passer d'une chaîne de caractères à un nombre sans à indiquer le type de la variable, c'est une force mais aussi une faiblesse, nous verrons pourquoi plus tard.

Qu'est ce qu'une variable ?

Comme nous l'avons vu juste avant, le Lua est un langage "non typé", ce qui signifie que vous n'avez pas besoin de déclarer le type de valeur

qui sera saisie dans votre variable.

Mais attention, il y aura toujours deux types de variables. Les variables locales et les variables globales.

La grande différence entre une variable locale et une variable globale sera sa "portée", une variable locale ne peut être utilisée que dans le bloc de code dans lequel elle est déclarée.

Par bloc de code, il faut comprendre : le fichier, la fonction, la boucle ou la structure conditionnelle (condition if, else, elseif) dans laquelle se trouve la variable.

Variable locale (local)

Exemple :

- Je crée un fichier "main.lua"
- J'y déclare une variable locale appelée "Text"
- Je crée un second fichier "main-2.lua"
- J'essaie de faire un "print" de ma variable "Text", la console renvoie "nil"

Elle renvoie "nil" parce que dans main-2.lua je n'ai jamais déclaré une variable nommée "Text".

Les variables locales ont également l'avantage d'être plus performantes que les variables globales, les variables locales seront détruites après leur utilisation, celle-ci étant temporaire pour stocker des informations dans votre bloc de code actuel.

Une variable locale ressemble à ceci :
Dans mon exemple, je vais attribuer "Hello World" à une variable locale appelée "Text".

```
local Text = 'Hello World';
```

Cette variable "Text" ne peut être utilisée que dans le fichier qui l'a déclarée.

Variable globale

Exemple :

- Je crée un fichier "main.lua"
- J'y déclare une variable locale appelée "Text"
- Je crée un second fichier "main-2.lua"
- J'essaie de faire un "print" de ma variable "Text", la console renvoie la valeur de "Text"

Cette fois, elle renvoie la bonne valeur, simplement parce que les variables globales sont accessibles partout, quel que soit l'endroit où elles sont déclarées.

Une variable globale ressemble à ceci :
Dans mon exemple, je vais attribuer la chaîne de caractères "Hello World" à une variable globale nommée "Text"

```
Text = 'Hello World';
```

Cette variable Text peut être utilisée dans n'importe quel bloc de code de notre projet.

L'un des inconvénients des variables globales est qu'elles restent stockées en mémoire afin d'être accessibles. Les coûts en performance dus aux variables globales peuvent varier, mais il est tout de même conseillé de les utiliser le moins possible, en utilisant au maximum les variables locales.

Attention, si vous faites des variables globales, leurs noms sont importants, il ne faut jamais déclarer deux fois le même nom.

Si vous déclarez une variable globale dans un fichier, et que vous déclarez cette même variable dans un second fichier, l'une va écraser l'autre.

Exemple :

- Je crée un fichier "main.lua"
- J'y déclare une variable locale appelée "Text"

- Je crée un second fichier "main-2.lua"
- J'y déclare une variable appelée "Text"
- J'essaie de faire un "print" de ma variable "Text", la console renvoie la valeur de "Text" mais celui de mon second fichier (main-2.lua)

En code cela donnerait ceci :

```
-- main.lua
Text = 'Coucou'

-- main-2.lua
require('main')
-- Cette petite ligne est obligatoire pour indiquer que nous avons besoin des "éléments"(variables, fonctions etc.) de mai

Text = 'Bonjour'
print(Text)

-- Resultat : 'Bonjour'
```

Que sont les fonctions et les arguments ?

Comme dans 95% des langages de développement, le Lua peut avoir des fonctions. Une fonction est une partie de votre code qui effectue une action lorsqu'elle est appelée.

Dans la programmation orientée objet, nous parlerons de méthode. La programmation orientée objet fera l'objet d'une future session.

Selon sa "portée" (locale ou globale), la fonction peut être appelée à l'intérieur ou à l'extérieur du bloc de code déclarant. Comme pour les variables, si votre fonction est locale ou globale, elle aura une certaine "portée".

Une fonction ressemble à ceci :
Fonction locale

```
local function superTest(argument_1)

end
```

Fonction globale

```
function superTest(argument_1)

end
```

Lorsque nous utiliserons notre fonction, nous lui enverrons des paramètres :

```
superTest(paramètre_1)
```

La fonction va donc traiter ces **paramètres** et ils doivent correspondre aux **arguments** de la fonction.
Ici, notre fonction superTest attend un argument appelé argument_1.

Notre fonction recevra alors des arguments qui seront traités par elle-même si nécessaire.

Quand **vous déclarez** une fonction, nous parlons d'**arguments**. Quand vous appelez votre fonction, nous parlons de **paramètres**.
Vous pouvez utiliser le mot "arguments" dans les deux cas, ce n'est pas une erreur.

Si **vous appelez** votre fonction et lui passez des **paramètres**, les **arguments** utilisés à l'intérieur de la fonction seront automatiquement locaux à la fonction et deviendront donc des variables locales.

Je vais imager tout cela en code :

```lua
local function superTest(chiffre)
  chiffre = chiffre + 1
  print(chiffre)
end

superTest(1)

-- Resultat : 2
```

Ici, ma fonction superTest est en attente d'un **argument**.
Je l'appelle donc en mettant comme **paramètre** (1), la fonction va alors traiter son **argument**, qui deviendra alors la **variable** "chiffre".

La **variable** "chiffre" sera toujours locale à la fonction superTest.

```lua
--[[ Début de mon bloc de code ]]--

local function superTest(chiffre)
    -- Un fois l'argument traité dans la fonction, il devient une variable locale

  chiffre = chiffre + 1
  -- Chiffre passe donc d'argument à variable locale, de manière automatique

  print(chiffre)
end

--[[ Fin de mon bloc de code ]]--

-- On appel donc notre fonction superTest et on lui passe le paramètre / l'argument (1)

superTest(1)
```

Vous pouvez donner un nombre illimité d'**arguments** à votre fonction
Pour cela, il existe *2 techniques*:

Soit vous déclarez tous vos arguments à la main afin de les traiter dans votre fonction.

```lua
local function superTest(arg_1, arg_2, arg_3, arg_4)
  arg_1 = arg_1 + 1
  arg_4 = arg_4 + 1
  print(arg_1, arg_2, arg_3, arg_4)
end

superTest(1, 2, 3, 4)
```

Dans cet exemple, je peux donc traiter mes **arguments** dans le cadre de ma fonction.

Si vous savez que vous n'aurez pas à traiter les **arguments** et que le nombre d'arguments peut varier, vous pouvez utiliser un autre moyen.

```lua
local function superTest(arg_1, ...)
  arg_1 = arg_1 + 1
  print(arg_1, ...)
end

superTest(1, 2, 3, 4)
```

Dans cet exemple, je peux traiter mon argument "arg_1", mais je ne pourrais pas traiter le reste au cas par cas.
Je dis que je ne pourrais pas le traiter au cas par cas, en fait je le peux en faisant "exploser" mon argument `*...*`

Je dis que je ne pourrais pas le traiter au cas par cas, en fait je le peux en faisant "exploser" mon argument `...` :

Je vais vous donner un exemple (pour les curieux) mais je ne le détaillerai pas au cours de cette session, il fera l'objet d'une analyse lors de la prochaine session.

```
local table = {};

local function superTest(...)
  for k, v in pairs(arg) do
    table[k] = v
  end

  table[3] = table[3]..' iThorgrim'

  print(table[3])
end

superTest('Bonjour', 'Salut', 'Heyio', 'Hello')

-- Resultat : 'Heyio iThorgrim'
```

J'espère que ce guide vous aura aidé à en apprendre un peu plus sur Lua, et j'espère vous avoir aidé.