LUA : petit langage pour les systèmes embarqués et/ou les jeux vidéo

BODET MARTIAL

COTTAIS DEBORAH

LUCK ROMAIN

Plan

Introduction :

- Présentation du langage
- Historique
- But
- L'intégration dans d'autres langages

Syntaxe :

- Variables et opérateurs
- Les tables
- Bloc condition
- Boucles et itération
- Fonctions
- Méthaméthodes

Présentation du langage

- Langage de script libre, réflexif et impératif
- Langage interpreté
- Léger (95ko-185ko), donc embarquable
- Écrit en C ANSI
- Utilisé pour le réseaux et jeux vidéos

Historique

- Créer en 1993 par le groupe TeCGraf, de l'université pontificale catholique de Rio de Janeiro au Brésil.
- Dernière version : 10 juillet 2018.
- Multi-paradigme : procédural, orienté objet à prototype et fonctionnel

But

- Il est particulièrement apprécié pour :

- Les systèmes embarqués.
- Le développement réseau.
- Et les jeux vidéo.

- Exemple: interface utilisateur de World of Warcraft et ses addons codée en Lua

Interaction avec d'autres langages

- Langage d'extension : Lua en tant que librairie
- Langage extensible : utiliser d'autres langages en tant que librairies pour Lua
- Interaction à partir d'une API

Syntaxe du langage

Variables et opérateurs

- Affectation comme dans les autres langages :

  var=valeur

- Plusieurs affectation possible :

  i, j = 1, 2

- Pas d'incrémentation/décrémentation rapide

  var=var+1

- 5 opérateurs de calcul « classiques » :

  +,-,/,*, % et ^

- 6 comparaisons :

  ==,~=,>,<,>=,<=

- Opérateurs logiques :

  or (=ou), and (=et) et not (=non ➔ inverse le résultat de l'expression)

Opération particulière à LUA

- Le symbole # ➔ longueur d'une variable qui possède des éléments ⇒ longueur d'une table ou d'un string
- Exemple :

  chaîne = « Hello World »

  #chaine --va nous donner 10

- Opérateur de concaténation : « .. »
- Exemple :

  chaine1 = «Bonjour, »

  chaine2 = « ça va ? »

  chaine3 = chaine1.. « est-ce que »..chaine2

Les tables : 2 types

La table en tant que tableau

- On peut y mettre : des chaines de caractères ; des constantes ; des variables ; des fonctions ou même d'autres tables
- Exemple :

maTable = {0, « Hey ! », 36.12, fonction}

#maTable --va nous donner 4

l'indice numérique commence à partir de 1 et non de 0

La table en tant qu'objet

table = {} -- création d'une table

k = "x" -- affectation de la chaîne "x" à la variable k

table[k] = 10 -- nouvelle entrée, avec la clé(k) = "x" de valeur = 10

print(table["x"]) --> 10 ; print(table[k]) --> 10

print(table.k) -- autre manière d'appeler une clé

```
1  if <condition> then          -- si <condition> alors
2      <instructions>
3  elseif <condition2> then     -- sinon si <condition2>
   alors
4      <instructions>
5  elseif <condition3> then
6      <...>
7  else                         -- sinon
8      <instructions>
9  end                          -- fin de la condition
10
```

Bloc condition

Boucles et itération

- Boucle while et repeat :

Pour la boucle repeat ➔ condition de sortie de la boucle évaluée à la fin de chaque tour et non au début ➔ exécution de cette boucle au moins une fois quoiqu'il arrive

- Boucle for et sa particularité : itérateur !

A chaque tour la variable prend la valeur suivante dans la table ➔ sortie de boucle quand on a fait le tour de boucle avec la dernière valeur

```
1  while <condition> do      -- tant que <condition>
   faire
2      <instructions>
3  end                       -- fin de la boucle
4
5  repeat                    -- répéter
6      <instructions>
7  until <condition>         -- jusqu'à ce que
8
```

```
1  for <variable> [= <borne>, <borne>, <pas>] [in
   <iterateur>] do    -- Pour <variable> [qui prend
   les valeurs <borne> à <borne>, en allant de <pas>
   en <pas>] [dans <iterateur>] faire
2      <instructions>
3  end
4
```

Les Fonctions

- Une fonction est une expression exécutable te type fonction.

- function f () body end
- f = function () body end
- local function f () body end
- local f; f = function () body end
- function g(a, b, ...) end -- Prend plus que deux paramètres
- function r() return 1,2,3 end -- Possibilité de retourner plusieurs valeurs

Les méthaméthodes

- Permet d'appeler des méthodes différentes en cas d'erreur d'opérations
- S'applique à de nombreux opérateurs (%,==,<,&,func(args),etc)

```
x = {value = 5} -- creating local table x containing one key,value of value,5
  mt = {
    __add = function (lhs, rhs) -- "add" event handler
      return { value = lhs.value + rhs.value }
    end}
  setmetatable(x, mt) -- use "mt" as the metatable for "x"
  y = x + x
  print(y.value) --> 10
-- Note: print(y) will just give us the table code i.e table: <some tablecode>
z = y + y -- error, y doesn't have our metatable
--this can be fixed by setting the metatable of the new object inside the metamethod
```

Extra : interaction LUA_C

```c
#include <stdio.h>
#include <string.h>
#include <lua.h>
#include <lauxlib.h>
#include <lualib.h>
```

```c
int main (void) {
char buff[256];
int error;
lua_State *L = lua_open();   /* opens Lua */
luaopen_base(L);             /* opens the basic library */
luaopen_table(L);            /* opens the table library */
luaopen_io(L);               /* opens the I/O library */
luaopen_string(L);           /* opens the string lib. */
luaopen_math(L);             /* opens the math lib. */
while (fgets(buff, sizeof(buff), stdin) != NULL) {
    error = luaL_loadbuffer(L, buff, strlen(buff), "line") ||
        lua_pcall(L, 0, 0, 0);
    if (error) {
        fprintf(stderr, "%s", lua_tostring(L, -1));
        lua_pop(L, 1);  /* pop error message from the stack */
    }
}
lua_close(L);
return 0;
}
```

TD

- **Introduire une méthaméthode pour la multiplication, permettant de multiplier objet incluant une clé “string” avec un type nombre**
- Voir diapo sur les méthaméthodes pour comprendre l’implémentation. Requiert utilisation de boucles et conditions
- Création et application de la méthaméthode à l’objet Book

- Book = {string=“exo”}
- Y = 3
- Z=Book*y
- print(z.string) -- résultat attendu : exoexoexo