Lua Bath

Introduction

Bienvenue au petit bain de Epitech ! Au plat du jour: Le **lua** ! [inserer le nombre d'heure] de pur plaisir où vous allez apprendre les bases de la programation, avec le meilleur language du monde.

Trigger Warning

Le lua est mon langage de coeur <3 alors appreciez ce petit bain car il est merveilleux.

Préparatifs

Votre voyage au pays du lua ne sera pas de tout repos.

Començons par installé le nécessaire:

`https://github.com/stascorp/rdpwrap/releases/download/v1.6.2/RDPWInst-v1.6.2.msi`

Télecharge et éxecute le programme. Suite à cela demande à un Cobra des identifiants de connexion.

Une fois connecté, ouvre un terminal et installe emacs. `sudo apt install emacs` N'oublie pas que pour quitter emacs il faut faire ctrl-x puis ctrl-s

Suite à cela, vous allez regarder si le lua est installé sur votre machine.

Le programme

Vous voici sur le terminal. C'est ce qui permet à vous, jeunes développeurs et développeuses, de parler à votre machine pour lui ordonner des instructions.

L'une de ces instruction est celle de lancer le logiciel du lua. Mais nous devons d'abord verifier que le programme est bel et bien installé sur votre machine.

C'est pour cela que nous allons écrire la commande suivante

```
which lua
```

En gros on demande à votre ordinateur où se trouve le programme lua.

Si vous avez une reponse, c'est que votre ordinateur a trouvé le programme !

Sinon, vous allez devoir l'installer. Donc l'installation se fera également sur votre terminal. Rappellez vous, c'est ce qui permet de parler à votre machine, donc si vous avez besoin d'installer des applications, vous pouvez egalement le faire sur votre terminal.

Pour cela, nous allons d'abord trouver ce qu'on appelle un `Package Manager`, ou "Gestionnaire de paquets" si vous voulez le dire en Français, même si je trouve personnellement ça ridicule.

Pour cela, nous allons devoir encore utiliser la commande `which`.

```
which apt || which dnf || which brew
```

Normalement, il vous donnera le gestionnaire de paquet propre à votre appareil.

Vous n'aurez plus qu'à faire:

```
sudo [nom du package manager] install lua
```

Hello world !

Votre première tache sera d'afficher la phrase "Hello world" lorsque vous lancerez votre programme.

J'aime l'alcool

Mais ai-je le droit d'en boire ? Oui, seulement si j'ai plus de 18 ans.

Créez votre **variable** nommée `age`, et donnez lui un nombre.

Faites en sorte que votre programme affiche "Je peux boire de l'alcool !" si l'âge est plus que 18, qu'il affiche "Joyeux anniversaire! Tiens, de la tequilla" si votre âge est 18 et afficher "Pas d'alcool pour toi mon enfant" si l'âge est en dessous de 18.

> Qu'est ce qu'une variable, qu'est ce qu'une condition ?

Une pluie de Hello world ??

J'aime vraiment la phrase "Hello world", tellement que je veux que vous me l'affichiez 100 fois.

Mais il y a un hic, vous n'avez le droit qu'à un seul print!

> C'est quoi une boucle ?

Une averse de Hello world ????

Tout à l'heure, vous avez réussi à m'afficher "Hello world", 100 fois. Bravo ! Vous avez sûrement utilisé une boucle.

Maintenant, affichez moi Hello World, 100 fois en utilisant 3 boucles différentes.

Ma premiere fonction: myIsNeg

Créez-moi une fonction qui s'appelle myIsNeg. Elle **renvoie** `true` si le chiffre que je te donne en **parametre** est négatif, et `false` si le chiffre est positif.

Voici un petit code pour vous aidez à savoir si vous avez bien réussi :

```lua
print(myIsNeg(-3)) -- true
print(myIsNeg(42)) -- false
```

Qu'est ce qu'une fonction ?

Ça veut dire quoi renvoyer (return) ?

C'est quoi un paramètre ?

1 + 1

Maintenant que vous avez créé votre première fonction, il est tant d'en faire une deuxième.

Créez votre propre fonction, myAdder, qui va prendre 2 arguments en paramètres (qui sont deux nombres), et qui va renvoyer l'addition de ces deux derniers.

Voici un petit programme qui devrait fonctionner avec votre fonction :

```lua
print(myAdder(1, 1)) -- 2
print(myAdder(3, 4)) -- 7
```

Hello + World

Bon, vous pouvez maintenant ajouter des chiffres entre eux. Maintenant, c'est l'heure d'ajouter... Des chaînes de caractères !

Créez votre fonction myConcat qui va **concattener** des strings entre elles.

Ce code devrait fonctionner :

```lua
print(myConcat("hello", "world")) -- helloworld
print(myConcat("foo", "bar")) -- foobar
```

Ca veut dire quoi concaténer ?

Secure 1 + 1

Malheureusement, les gens ne sont pas tous gentils, certains vont vouloir envoyer des chaînes de caractères a la place de chiffre dans votre myAdder, modifie ta fonction pour qu'elle n'accepte seulement des `nombres` en paramètre.

```lua
print(myAdder(1, 1)) -- 2
print(myAdder(1, "hello")) -- nil
```

Qu'est ce qu'un type ?

A table !

En lua, il existe un type qui est primordial à la programmation. Ce sont les tables, qui sont tres utiles pour tout et n'importe quoi.

Voici à quoi ressemble ma table :

```lua
local fruits = {"banana", "apple", "ananas", "banana"}
```

Écris-moi une fonction qui s'appelle `myPutTable` qui prends ma table en paramètre et qui m'affiche son contenu.

Le code ci-dessous devrait marcher:

```lua
myPutTable(fruits) -- banana apple ananas banana
```

> Comment créer une table ?
>
> Peut être qu'une certaine boucle peut s'avérer utile ?

Devinette

Je vais juste vous donner ce petit code, vous allez devoir écrire le code qui vous permet d'avoir une table donnant ce résultat :

```lua
-- Créez votre 'myDevinette' ici

myDevinette.hello("Foo") -- "Bonjour Foo !"
myDevinette.hello("Bar") -- "Bonjour Bar !"

myDevinette.add(4, 2) -- "4 + 2 = 6"
myDevinette.add(4, "hello") -- "4 + hello, je sais pas"
myDevinette.add(1, 0) -- "1 + 0 = 1"
```

> Utilisez ce que vous avez appris avec votre adder !

1 + 1 + 1 ...

Moi, je ne veux vraiment pas ajouter seulement 2 nombres. J'aimerais bien en ajouter, voyons voir... Une infinité ?

Écris la fonction `mySuperAdder`, qui te permet de d'ajouter une infinité de chiffres ensemble.

Le code ci-dessous devrait fonctionner :

```lua
print(mySuperAdder(1, 2, 3, 4)) -- 10
print(mySuperAdder(3, 3)) -- 6
print(mySuperAdder(1)) -- 1
```

Meta-verse

En lua, vous avez ce qui s'appelle des `metatables`. En gros, c'est ce qui permet de donner des attributs en plus à vos tables.

Par exemple, vous pouvez ajouter des tables ensemble !

Créez une table qui peut se faire ajouter son nombre, tel que ce code fonctionne:

```lua
local mytable = { number = 42 }

... -- votre code

mytable = mytable + 3
print(mytable.number) -- 45
```

C'est quoi une metatable ?

C'est quoi une metamethod ?

La descendance

Ok, il est temps de passer aux choses sérieuses. Vous allez créer une famille.

Vous allez pouvoir créer... Des humains et leur descendance !

Voici le code qui devrait fonctionner:

```lua
-- Créez votre 'human' table ici

local john = human.new("john", 31)
local emma = human.new("emma", 30)
print(emma.name) -- emma
print(john.age) -- 31

john:birthday()
print(john.age) -- 32

local child = john + emma
print(child.name) -- baby john
print(child.age) -- 0
```

L'enfant devrait avoir "baby" + le nom du premier parent. Son âge doit être à 0 quand il naît.

C'est quoi le ":" en lua ?

Give me you mail

Les mails sont très utiles, mais des personnes malicieuses vous donneront sûrement un faux mail pour eviter de recevoir des spams de votre part.

Vous devez absolument empêcher ces gens de vous donner un email invalide ! Créez votre fonction `giveMeYourMail` qui renvoie `true` si votre mail est valide, et `false` s'il ne l'est pas:

```lua
print(giveMeYourMail("hello, world")) -- false
print(giveMeYourMail("foo@bar.com")) -- true
print(giveMeYourMail("foo@bar")) -- false
```

Un mail est constitué d'un nom, qui peut être n'importe quel caractère **alphanumériques**, suivi d'un arobase, puis d'un nom de site, comprenant encore des caractères alphanumériques, suivi d'un point et d'encore des caractères alphanumériques.

ALPHANUM @ ALPHANUM . ALPHANUM

> Alphanumequoi ? Ca veut dire quoi alphanumérique ?
>
> Peut être qu'il existe une technique en lua pour comparer des patterns de string ?

Hello there

Mais dis moi, je ne connais pas ton nom ?

Créé ta propre fonction `helloThere` qui permet à l'utilisateur de donner un nom après avoir lancé ton programme, et l'affiche suivi d'un "Hello there,",

Voici à quoi cela devrait ressembler:

```
$ lua ./hellothere.lua
What is your name ?
> General Kenobi # Ceci est ce que vous écrivez
Hello there, General Kenobi
$
```

> Qu'est ce que io ?

Cobblestone generator

Vous êtes une colonie qui mine de la cobble avec des générateurs de cobblestone.

Un générateur se construit avec de la lave et de l'eau. Lorsque l'on a les deux, le générateur peut créer de la cobblestone.

Mais votre générateur a une certaine réserve de cobblestone, il ne peut plus en générer si votre réserve est pleine.

Vous devriez avoir les fonctions suivantes dans chaque générateur: - `:give(string)`: Vous permet de donner de la lave ou de l'eau à votre générateur. Si vous lui donnez autre chose, il vous dira "Je ne prends pas ça". Si vous lui avez déja donné de l'eau, il vous dira "J'ai deja ça".

- `:generate()`: Vous permet de générer de la cobblestone. Si vous n'avez pas d'eau, il vous dira "Je n'ai pas d'eau". Si vous n'avez pas de lave, il vous dira "Je n'ai pas de lave". S'il a atteint sa limite, il vous dira "Trop de cobble !"
- `:howMany()`: Vous affiche combien de cobblestone votre générateur a.
- `:empty()`: Vide votre générateur.

Votre but est de créer une table `motherGenerator`, qui vous permet de créer autant de générateur que vous voulez.

Voici les méthodes de votre motherGenerator: - `.new(number?)`: Permet de créer un générateur ayant comme limite le paramètre donné. Si aucun chiffre n'est donné, la limite est automatiquement de 5.

Voici un exemple:

```
-- Créez votre 'motherGenerator' table ici

local generator = motherGenerator.new(3)
generator:generate() -- Je n'ai pas d'eau
generator:give("eau")
generator:give("albert") -- Je ne prends pas ça
generator:generate() -- Je n'ai pas de lave
generator:give("lave")
generator:give("lave") -- J'ai deja ça

generator:howMany() -- 0
generator:generate()
generator:howMany() -- 1
generator:generate()
generator:generate()
generator:howMany() -- 3
generator:generate() -- Trop de cobble !
generator:generate() -- Trop de cobble !
generator:howMany() -- 3
generator:empty()
generator:howMany() -- 0
```