
Comencemos...

1.Breve historia

Lua es un lenguaje de scripting multiplataforma derivado de C++, es muy ligero y su semántica es muy similar a otros lenguajes de scripting, por lo que lo pueden entender muy rápidamente. Fue creado en 1993 en una universidad brasileña. En la actualidad (desde hace algunos años), lua se utiliza como soporte para otras aplicaciones gracias a su simplicidad, principalmente sistemas de bases de datos, o "scripts" para juegos (Por ejemplo, el conocidísimo WoW [World of Warcraft]).

2.Enlaces importantes

Lo primero que van a tener que hacer es descargárselo. Aquí tienen la versión más reciente, **5.1 para Windows**:
http://luaforwindows.googlecode.com/files/LuaForWindows_v5.1.4-40.exe

También acá subí el **srlua**, una herramienta para crear .exe standalone de sus scripts:
http://proyectosmatu.webcindario.com/srlua.zip

Desde la página oficial de LUA, pueden testear sus programas:
http://www.lua.org/demo.html

Y les recomiendo bajarse el Notepad++, no sólo para LUA, se los recomiendo para tenerlo ya que es excelente para cualquier lenguaje:
http://descargar.portalprogramas.com/Notepad-Portable.html

3.Primer paso

Una vez que lo tienen, lo instalan y abren LUA (Command Line). Ésta es la consola, donde probarán lo básico.

Hello World
Bueno, como ya sabían, vamos a empezar por hacer que la consola nos devuelva el mensaje "Hola mundo". Para eso vamos a utilizar la **sentencia print**. Ésta va de una forma muy simple: print (*mensaje*), y automáticamente se devolverá *mensaje* en la pantalla. Los paréntesis no son obligatorios generalmente, pero pueden ocurrir errores (en especial entre versiones diferentes) cuando se quiere imprimir el valor de una variable por ejemplo, por lo que es preferible usarlos.

Ejercicio: Hacer que diga Hola Mundo

4.Comentarios

Los comentarios se colocan con el símbolo de comentarios **--**. Todo lo que siga detrás de éste, será tomado como comentario por lo que el programa no lo tendrá en cuenta. Están en el código sólo para agregar anotaciones que guiarán a quien lo lea (sea el creador del programa o no), harán el código más ordenado, profesional y entendible.
Si queremos hacer comentarios de varias líneas, podemos utilizar --[[para comenzar, y]] para terminar. Ejemplo:

Código:
```
--[e esto va a ser tomado en cuenta
]]
[ Esto es un ejemplo
de un comentario con varias líneas
para el tutorial de LUA
Nada d
```

5.Variables

Las muy conocidas variables, son palabras que contienen un valor. Éstas las utilizaremos a lo largo del código de cualquier programa, por lo que son muy importantes.
Por empezar, no hay que definir el tipo de variable, sino que se "autodefine" según los datos que le asignemos.

Declarando variables

Las variables se crean mediante un nombre, el símbolo de asignación = (igual) y la información a asignar. El nombre debe ser descriptivo para que sea más comprensivo, y tampoco podemos usar palabras reservadas para este.
Entonces, para declarar una variable escribimos:
nombre = información. Por ejemplo: nombre = "Carlitos".

Ejercicio: Probar declarar variables en la línea de comandos y luego escribir el nombre de la variable sola para comprobar que devuelve.

Como les dije previamente, el tipo de la variable va a depender de lo que le ingresemos. Noten que en el ejemplo anterior, entrecomillé a Carlitos. Esto es porque las cadenas (string) van entre comillas dobles (" "). Los números, sean enteros, con coma (float, o de coma flotante), reales, etc; no van entre comillas, si los ponemos entre comillas, entonces serán cadenas por más de que sólo contenga números.

a)Cadenas

El tipo string se forma de todo tipo de caracteres: letras, números y símbolos. Se caracteriza por ir entre comillas. Una forma de usarlos es mediante la sentencia print. Por ejemplo:
Código:

```
bienvenida = "Hola Gente"
print (bienvenida)
```

Las cadenas se pueden formar con otras variables. Para hacer esto tenemos que **concatenar**. El símbolo de concatenación de lua es: **..**
Por ejemplo, supongamos que en una variable tengo la cantidad de alumnos hombres de una clase y en otra las mujeres. Con una cadena quiero decir "La clase tiene X hombres, y N mujeres". Entonces quedaría así:
Código:

```
cmujeres = 17
chombres = 11
mensaje = "La clase tiene "..chombres.." hombres, y "..cmujeres.." mujeres."
print (mensaje)
```

Como ven, las variables chombres y cmujeres van entre .., para concatenarlos en la cadena. Ademas se deja un espacio, ya que si no se dejara el espacio la devolución sería:
La clase tiene 11 hombres, y 17 mujeres.

Ejercicio: Comprobar que ocurre si se colocan las variables solas sin concatenarlas (es decir quitando los "..").

b)Números

Los números se utilizarán principal y obviamente para cálculos, y para recordar datos numéricos y mostrarlos cuando sea necesario. Los cálculos se realizan mediante operadores, los principales: +(suma); -(resta); *(multiplicación); /(división); ^n (exponenciar a n). Las operaciones se pueden expresar como cálculos combinados, es decir, se pueden utilizar los paréntesis, que hacen el código más ordenado.
Así, como podemos sumar número normalmente, también podemos sumar las variables que los contienen.

Con el ejemplo previo, podemos mostrar la cantidad total de alumnos de la clase:

Código:

Ejercicios: *hacer un programa que imprima el mensaje "El colegio tiene X alumnos", sabiendo que tiene 3 años (clases): El primer año tiene el total del ejemplo anterior, el total del segundo es el doble del primero, y el tercero la mitad del primero. Usar variables y print ().
*Comprobar que ocurre si se intenta sumar una variable cadena con una variable número. Ej.:
nombre = "Juan"
numero = 8
nombre+numero

c)Booleanas

Las booleanas representan un dato que puede ser verdadero (true) o falso (false). También podríamos usar los números 0 y 1, o cadenas como "si" y "no" para cumplir la misma función, pero las booleanas quedan más profesionales, además de ser las que menos memoria consumen. Se asignan igual que el resto: encendido = true, encendido = false.

4)Nil

El valor nil corresponde justamente a un valor nulo, es decir no es nada. Si se lo asignamos a una variable (chombres = nil, por ejemplo); la destruímos.

Alcance

a)Globales

Las variables además poseen alcance. Las variables como fueron declaradas previamente son globales, esto quiere decir que podrán ser usadas en cualquier parte del código, existirá en cualquier estructura que se use, y si se cambia su valor se cambiará en el código completo.
b)Locales
En cambio, si a la declaración de la variable le colocamos la palabra reservada **local**, la variable se creará sólo para esa estructura, es decir que si la declaramos dentro de una función, la variable existirá en esa función, pero si la llamamos fuera de la función, no existirá, y si existe es porque la creamos previamente, y seguramente tendrá otro valor, ya que aunque tengan el mismo nombre son dos variables distintas. Aún aunque sean locales, es recomendable no usar los mismos nombres en las funciones.

6.Funciones

Definiendo funciones

Una función es un bloque de código que al ser llamado, hace que el flujo del programa se detenga, se dirija a la función, ejecute sus sentencias, y regrese al flujo normal. Esto nos permite evitar las repeticiones, acortar el código, hacerlo más simple, más profesional, más ligero y más entendible.
Las funciones se definen con la palabra reservada Function, el nombre de la función, y sus argumentos entre paréntesis, separados por comas. El nombre de las funciones al igual que en el de las variables, debe ser descriptivo para ser entendible.
Los argumentos serán variables locales que ingrese el usuario al llamar la función, y que luego la función los maneje y los utilice para realizar una determinada orden.
Una vez escrito el conjunto de órdenes y sentencias, cerramos la función con **End**.

Return

Pero, antes de cerrar la función, podemos usar ***return*** *DEVOLUCIÓN*. Esto hace, que la función no se utilice para modificar algo, sino para obtener un dato que luego será manipulado. Si no usamos

return, entonces la función será considerada como un procedimiento que hará cambios en el código. Por ejemplo, podemos hacer que dentro de una función se le cambie el valor a 5 variables, por lo que no es necesario usar return. Mientras que si una función se utilizara para que a una variable se le asigne el valor de x+y; se utilizaría return x+y, y al llamarla se pondría cuenta = hacercuenta(10,8)

Por ejemplo, supongamos que yo quiero hacer un programa que haga una determinada suseción de operaciones de unos gastos que voy a hacer, que sería muy repetitivo de no ser que no hiciera una función para llamarla cada vez que lo necesite. La función tomaría dos números, los sumaría entre sí, al resultado le sumaría 10 y luego lo multiplicaría por 3. Por lo tanto, mi programa sería:

```
--Declaramos la función, como "cuentas", con los argumentos a y b
Function cuentas(a,b)
  local primercuenta = a+b
  --Fíjense que utilizo una variable local, será sólo existente en ésta función
  local segundacuenta = (primercuenta+10)*3
  --Noten que utilizo paréntesis, sino multiplica 10*3 y luego se lo suma a primercuenta
  return segundacuenta
End
--Ahora llamo a la función, asignándole el valor de devolución (el return, osea
--segunda cuenta) a una variable que luego la imprimiré

vtotal = cuentas(tonumber("7a"),8)
print ("El total a pagar será de "..vtotal.." pesos")
```

Ejercicios: *Probar llamar la función cuentas cambiando los argumentos y viendo que devuelve. *Lograr imprimir la función cuentas(a,b) directamente, sin asignársela a una variable previamente. *Probar llamar a la función en la línea de comandos directamente, sin ingresarla dentro de print, y ver que ocurre.

Funciones Predefinidas

Las funciones predefinidas son aquellas que ya vienen definidas por LUA, ya existen aunque no las veamos y las podemos utilizar sin necesidad de definirlas. Estas son una gran cantidad y se dividen según la función que cumplen, por ejemplo están las math que realizan cálculos matemáticos (funciones trigonométricas, raíz cuadrada, etc) os que devuelve datos del sistema (como la fecha, la hora, eliminar archivos, etc). Otras no pertenecen a ningún grupo, incluso print es una función predefinida, o acaso no notaron los paréntesis.
Una muy útil es dofile(ruta), que permite incluir un script o archivo de texto en nuestro programa. Es muy interesante para realizar bases de datos.
Las funciones tonumber(variable) y tostring(variable), como su nombre lo indica, permiten cambiar el tipo de la variable que le ingresemos.
Para más documentación sobre las funciones predefinidas (son bastantes pero no todas del todo necesario, por eso si realmente necesitan una búsquenla o pídanme ayuda), en Google, la mayoría está en inglés pero es entendible.

Ejercicio: Realizar un programa corto pero útil, que utilice variables y una (y si es posible varias) funciones, y por lo menos 2 funciones predefinidas; que esté bien ordenado, con nombres claros y comentar todo lo posible. Puede estar basado en el ejemplo del colegio, de los pagos, o lo que quieran.

a)io.read()

Como les dije, hay muchas funciones predefinidas, todas muy útiles; pero dentro de ellas voy a resaltar algunas. Una de estas es io.read (similar a raw_input). Lo que hace es pedirle al usuario que ingrese algo y presione enter para continuar. Lo que escriba se le puede asignar, por ejemplo, a una variable...de esta manera:

[Ejemplo]
Código:

```
>nombre = io.read()
_                    El "_" representa el puntero que aparece para que el usuario
ingrese
algo, supongamos "Marta"
>print(nombre)
Marta
```

b)os.execute("pause")

Otra es os.execute("pause"), que es el equivalente a system("pause"). Lo que hace es pausar el programa y no continuar hasta que no se aprete una tecla, mostrando en pantalla el clásico mensaje "Presione una tecla para continuar..."

Parámetros extra y funciones vararg

Lua tiene una propiedad característica, en la que al llamar una función, si le colocamos parametros de más, no los tomará en cuenta. Esto se relaciona con las*funciones vararg*. Vararg proviene de "variable arguments" (probablemente). Dichas funciones, son iguales al resto sólo que en sus argumentos contiene una*expresión vararg*: **...** (tres puntos). Quiere decir que estas funciones no definen sus argumentos, sino que toman todos los extras y los administran para hacer ciertas devoluciones.

Les voy a explicar con ejemplos de funciones. Supongamos que nosotros tenemos la función sumar(a,b) (y que en el cuerpo sume a+b). Por lo que les dije primero, si nosotros llamamos sumar(1,4,6), va a sumar 1+4 (a+b) y devolverá 5, sin importar el 6. Si nosotros llamamos sumar(1), entonces devolverá error al intentar sumar 1 con nil (ya que al no valorar b, lo toma como nil).
Si en cambio definimos una función cuentas(a,b,...), estoy utilizando "...". Si lo llamamos como cuentas(1,5); tomará a y b como number, y a "..." lo descartará ya que no es necesario. Mientras que si llamamos como cuentas(1,5,6,7,8) tomará todos los numeros que ingresamos como number, gracias al vararg.

También podemos utilizar el múltiple return, colocándolos entre comas. Ejemplo:

```
function contar()
        return 1,2,3,4
end
```

Si esa función la usamos para llamar a cuentas: cuentas(contar()), sería como llamarla así: contar(1,2,3,4), por lo que tomará esos 4 valores como number.

7-Condicional

El condicional IF, es una estructura que permite colocar un cuerpo con un conjunto de órdenes que se ejecutarán sólo si la condición establecida es verdadera. Si conocen lo básico de cualquier lenguaje de programación actual sabrán muy bien de que hablo, sino, lo entenderán muy fácilmente.

IF, AND y OR

Es tan simple como esto: (Ejemplo) Si está lloviendo, entonces llevar un paraguas. La estructura es
If condición Then
Órdenes
End

Si nos olvidamos de alguna de éstas palabras (if, then, o end) o la colocamos en posición incorrecta, como en cualquier otra estructura obtendremos un error.

Además contamos con dos reservadas más, OR y AND. El and nos permitirá usar varias condiciones (Ejemplo en castellano: Si es día y es fin de semana entonces ir de picnic [IF es de dia AND es fin de semana THEN ir de picnic). En este caso se deben cumplir **las dos condiciones**, si se cumple una y la otra no entonces no es válido.
En cambio, si utilizamos OR, puede que se cumpla una o la otra, o ambas. (Si es fin de semana O es feriado, entonces no ir al colegio [IF es fin de semana OR es feriado THEN no ir al colegio])

Si el or se utiliza dentro de una condición sin iniciar una condición distinta, es decir en la misma condición, ejemplo: if calzado == "zapatilla" or "zapato" or "sandalia" then; entonces la condición va a observar la primer cadena, y si es correcta la va a tomar, sino va a proseguir con la siguiente y así hasta encontrar una correcta. Si no lo hace, entonces la condición es falsa.

Si la sentencia es una sola orden, no un conjunto, entonces se puede hacer en una sola linea IF condición THEN sentencia, sin necesidad de ponerlo en vertical.
Si la condición es una booleana, entonces no es necesario poner IF variable==TRUE THEN, se puede poner IF variable THEN, entendiéndose que es TRUE.

Signos de Comparación

Como les expliqué recién, IF se forman de IF condición THEN. Esa condición debe ser una expresión que solo permitirá ejecutar la sentencia si se cumple. Esta expresión se forma con caracteres especiales para crear una condición. Ellos son:

> (mayor que)
< (menor que)
== (igual a. RECUERDEN QUE el signo de comparación "igual que" es ==. Si colocamos un solo igual (=), estaremos utilizando el signo de asignación, para crear o asignar una variable)
>= (mayor o igual que. EL ORDEN DEBE SER > y luego =, Y SOLO EN ESE ORDEN)
<= (menor o igual que)
~= (desigual a)

Igualar a Nil: si igualamos a nil (==nil), entonces lo que haremos es preguntar si sierta variable existe o no.
if variable==nil (variable no existe)
if not variable==nil (variable existe)
if variable~=nil (variable existe)

NOT

Si a nuestro condicional le agregamos un Not detrás, entonces la condición debe ser incorrecta para que se pueda ejecutar la sentencia. Para que se entienda: Si no es fin de semana entonces ir al colegio (If not fin de semana then ir al colegio).
Con las booleanas, si colocamos IF not variable THEN, se entiende que queremos decir IF variable==false then. Cualquiera de las dos maneras es correcta, como les explicaba anteriormente.

Else y Else If

Else (que se puede traducir como "sino", "en cambio"), como la palabra lo dice, nos permite ejecutar un conjunto de órdenes en caso de que ocurra algo distinto a la condición sugerida. No es obligatorio, pero si muy útil. Su sintáxis es muy similar a muchos otros lenguajes de programación. Como vengo explicándoles "con lenguaje natural", voy a hacerlo con el Else para que lo comprendan bien.

SI está lloviendo ENTONCES (If...Then)
llevar un paraguas
SINO (Else)
dejarlo en casa

Pero, también, sino se cumple la condición principal, podemos establecer otra condición, si esa no se cumple podemos establecer otra, y así hasta que una se cumpla o ninguna lo haga, y siga de largo.
Para hacer esto podemos poner:
IF condición principal THEN
sentencia
ELSE IF otra condición THEN
sentencia
ELSE IF otra condición mas THEN
sentencia
ELSE
sentencia

Como verán termina, con "ELSE sentencia".
Ejercicio: Comprender que hace ese "ELSE sentencia" final.

Ahora, prosigan con los bucles. Allí podrán ver pequeños ejemplos de la utilización de IF y ELSE.

8-Bucles

Los bucles son estructuras que nos ahorrarán **muchas** iteraciones. Es necesario conocerlos muy bien para saber utilizarlos debido a que son muy recurrentes y el no conocerlos puede llevarlos a hacer códigos extremadamente raros, largos y poco vistosos, y aún menos entendibles.

For...do

Los bucles for son un bloque que al terminar de ejecutarse sus órdenes, en vez de continuar con el flujo del programa, volverá al inicio del bucle y repetirá las órdenes. Ésta repetición se realizará las veces que el usuario lo indique. La repetición puede ser un número directo o puede ser una variable, por lo que, claramente, se pueden crear funciones donde el usuario ingrese el rango del bucle. Los for se usan de la siguiente forma: for variable = inicio,final do.

Vamos por partes, primero comenzamos por for(palabra reservada), luego seguimos por variable. ¿Qué es eso de variable? quiere decir que ingresaremos un nombre de variable (local) utilizada para el bucle que luego será destruída. Cada vez que el bucle se repita el valor de la variable aumentará en 1, es decir que si el rango del bucle es de 1 a 5, entonces el valor de la variable al principio será 1, luego 2, 3, 4 y por último 5.
inicio y final determinarán el rango, por ejemplo: 1,5 ; 1,10; 6,45; etc.
Finalmente do. Luego debajo de eso colocaremos el conjunto de órdenes a ejecutar, y debajo end para terminar.
Lo más interesante de esto, es ciertamente la variable. Ésta será la base del for, y la podemos manipular para incluirla en la sentencia a ejecutar. Por ejemplo, si yo quiero "contar ovejas", y no detenerme hasta no haber contado 12, podría hacer esto:

Código:
```
for numoveja = 1,12 do
--Vamos a hacerlo interesante, con if
if numoveja==1 then --Si el número es 1 entonces dice "Veo una oveja"
print ("Veo una oveja")
 else --En cambio, si son mas de 1 dice "Veo X ovejas", sino queda feo
print ("Veo "..numoveja.." ovejas")
end
end
print("No veo más ovejas :(")
```

Espero que ahora lo hayan entendido mejor.

Ejercicio: Realizar un for que cuente hasta 20, pero de 2 en 2 comenzando por 2 (imprimiendo con print:
2
4
6
8
10
etc...)

While...do

Bueno, este bucle repetirá un bloque mientras una expresión sea verdadera. Por ejemplo, si le decimos que repita las órdenes hasta que variable<=5 (variable sea menor o igual que 5), y dentro de las órdenes colocamos variable = variable+1 (siendo que variable empieza como = 1), el bloque se repetirá 5 veces, ya que irá sumando su valor hasta que en un momento variable será mayor que 5 y la expresión será falsa; en ese momento el bucle terminará y el programa seguirá su flujo normal.
En este caso, haremos una cuenta regresiva...

Código:
```
gregresiva = 10
while gregresiva>=0 do
If gregresiva>0 then
print (gregresiva)
 else
print ("Feliz Navidad!")
end
```

```
end
```
Devolución: 10
9
8
7
6
5
4
3
2
1
Feliz Navidad!

Ejercicio: Buscar sobre el bucle repeat...until

9-Arreglos y Tablas

Arreglos

Aveces, necesitamos una misma variable para varias cosas distintas. Por ejemplo, una variable que contenga el color de pelo de una persona podría ser colorpelo = rubio, pero si es entre 25 personas...una forma podría ser colorpelo1 = rubio, colorpelo2 = castaño, etc... y así con las 25; pero hay una forma de hacer esto más compacto y maleable. Los arreglos (array) son los indicados en este caso. Simplemente creamos una variable, y entre corchetes ([]) su índice. El índice indicara que número es. Por ejemplo, si queremos saber el color de pelo de Carlos, quien es el número 8 escribiremos: colorpelo[8], y nos devolvera el color que contenga. Para crear un arreglo, declaramos una variable sólo que su valor irá entre llaves, por ejemplo colorpelo = {}. Y a partir de ahí podemos crear más, colorpelo[1] = "colorado".
Por supuesto, podemos hacer una función simple para crear arreglos de forma rápida.
La función sería más o menos así:

Código:
```
function creararreglo(n) --para N colocar el tamaño del arreglo (con el ejemplo, 25)
           arreglo = {}
           for i = 1,n do
                      arreglo[i] = 0
           end
           return array
end

--llamando a la función
colorpelo = creararreglo(25)
```

Tablas

Pero hay veces, que necesitamos varias variables por persona. Por ejemplo, además del color de pelo, talvez necesitamos el color de ojos y la altura. Es lógico utilizar tres arreglos por persona, colordepelo[1] = "castaño", colordeojos[1] = "marrones", altura[1] = 180. Pero hay casos en los que se necesitarían muchos arreglos. Entonces las tablas nos pueden facilitar esto, sean 10 arreglos o 3 como el ejemplo que les di, es recomendable usar las tablas. Se crean de la misma forma (persona = {}), sólo que dentro de las llaves ({}) irán directamente las variables que vamos a guardar, es decir: persona = {colordepelo = "",colordeojos = "",altura = 0}. Más adelante podemos modificar las

variables que contenga la tabla. Para usarlos, como en los arreglos, colocaremos persona[INDICE].variable. Ej, para definir: persona[8].altura = 174. y para imprimir simplemente persona[8].altura.1-Breve historia

---

Comencemos...

1.Breve historia

Lua es un lenguaje de scripting multiplataforma derivado de C++, es muy ligero y su semántica es muy similar a otros lenguajes de scripting, por lo que lo pueden entender muy rápidamente. Fue creado en 1993 en una universidad brasileña. En la actualidad (desde hace algunos años), lua se utiliza como soporte para otras aplicaciones gracias a su simplicidad, principalmente sistemas de bases de datos, o "scripts" para juegos (Por ejemplo, el conocidísimo WoW [World of Warcraft]).

2.Enlaces importantes

Lo primero que van a tener que hacer es descargárselo. Aquí tienen la versión más reciente, **5.1 para Windows**:
http://luaforwindows.googlecode.com/files/LuaForWindows_v5.1.4-40.exe

También acá subí el **srlua**, una herramienta para crear .exe standalone de sus scripts:
http://proyectosmatu.webcindario.com/srlua.zip

Desde la página oficial de LUA, pueden testear sus programas:
http://www.lua.org/demo.html

Y les recomiendo bajarse el Notepad++, no sólo para LUA, se los recomiendo para tenerlo ya que es excelente para cualquier lenguaje:
http://descargar.portalprogramas.com/Notepad-Portable.html

3.Primer paso

Una vez que lo tienen, lo instalan y abren LUA (Command Line). Ésta es la consola, donde probarán lo básico.

Hello World
Bueno, como ya sabían, vamos a empezar por hacer que la consola nos devuelva el mensaje "Hola mundo". Para eso vamos a utilizar la **sentencia print**. Ésta va de una forma muy simple: print (*mensaje*), y automáticamente se devolverá *mensaje* en la pantalla. Los paréntesis no son obligatorios generalmente, pero pueden ocurrir errores (en especial entre versiones diferentes) cuando se quiere imprimir el valor de una variable por ejemplo, por lo que es preferible usarlos.

Ejercicio: Hacer que diga Hola Mundo

4.Comentarios

Los comentarios se colocan con el símbolo de comentarios **--**. Todo lo que siga detrás de éste, será tomado como comentario por lo que el programa no lo tendrá en cuenta. Están en el código sólo para agregar anotaciones que guiarán a quien lo lea (sea el creador del programa o no), harán el código más ordenado, profesional y entendible.
Si queremos hacer comentarios de varias líneas, podemos utilizar --[[para comenzar, y]] para terminar. Ejemplo:

Código:
```
--[[ Esto es un ejemplo
de un comentario con varias líneas
para el tutorial de LUA
Nada de esto va a ser tomado en cuenta
]]
```

5.Variables

Las muy conocidas variables, son palabras que contienen un valor. Éstas las utilizaremos a lo largo del código de cualquier programa, por lo que son muy importantes.
Por empezar, no hay que definir el tipo de variable, sino que se "autodefine" según los datos que le asignemos.

Declarando variables

Las variables se crean mediante un nombre, el símbolo de asignación = (igual) y la información a asignar. El nombre debe ser descriptivo para que sea más comprensivo, y tampoco podemos usar palabras reservadas para este.
Entonces, para declarar una variable escribimos:

nombre = información. Por ejemplo: nombre = "Carlitos".

Ejercicio: Probar declarar variables en la línea de comandos y luego escribir el nombre de la variable sola para comprobar que devuelve.

Como les dije previamente, el tipo de la variable va a depender de lo que le ingresemos. Noten que en el ejemplo anterior, entrecomillé a Carlitos. Esto es porque las cadenas (string) van entre comillas dobles (" "). Los números, sean enteros, con coma (float, o de coma flotante), reales, etc; no van entre comillas, si los ponemos entre comillas, entonces serán cadenas por más de que sólo contenga números.

a)Cadenas

El tipo string se forma de todo tipo de caracteres: letras, números y símbolos. Se caracteriza por ir entre comillas. Una forma de usarlos es mediante la sentencia print. Por ejemplo:
Código:

```
bienvenida = "Hola Gente"
print (bienvenida)
```

Las cadenas se pueden formar con otras variables. Para hacer esto tenemos que **concatenar**. El símbolo de concatenación de lua es: **..**
Por ejemplo, supongamos que en una variable tengo la cantidad de alumnos hombres de una clase y en otra las mujeres. Con una cadena quiero decir "La clase tiene X hombres, y N mujeres". Entonces quedaría así:
Código:

```
cmujeres = 17
chombres = 11
mensaje = "La clase tiene "..chombres.." hombres, y "..cmujeres.." mujeres."
print (mensaje)
```

Como ven, las variables chombres y cmujeres van entre .., para concatenarlos en la cadena. Ademas se deja un espacio, ya que si no se dejara el espacio la devolución sería:
La clase tiene11hombres, y17mujeres.

Ejercicio: Comprobar que ocurre si se colocan las variables solas sin concatenarlas (es decir quitando los "..").

b)Números

Los números se utilizarán principal y obviamente para cálculos, y para recordar datos numéricos y mostrarlos cuando sea necesario. Los cálculos se realizan mediante operadores, los principales: +(suma); -(resta); *(multiplicación); /(división); ^n (exponenciar a n). Las operaciones se pueden expresar como cálculos combinados, es decir, se pueden utilizar los paréntesis, que hacen el código más ordenado.
Así, como podemos sumar número normalmente, también podemos sumar las variables que los contienen.

Con el ejemplo previo, podemos mostrar la cantidad total de alumnos de la clase:

Código:

```
cmujeres = 17
chombres = 11
```

```
ctotal = cmujeres+chombres
print ("La clase tiene "..ctotal)
```

Ejercicios: *hacer un programa que imprima el mensaje "El colegio tiene X alumnos", sabiendo que tiene 3 años (clases): El primer año tiene el total del ejemplo anterior, el total del segundo es el doble del primero, y el tercero la mitad del primero. Usar variables y print ().
*Comprobar que ocurre si se intenta sumar una variable cadena con una variable número. Ej.:
nombre = "Juan"
numero = 8
nombre+numero

c)Booleanas

Las booleanas representan un dato que puede ser verdadero (true) o falso (false). También podríamos usar los números 0 y 1, o cadenas como "si" y "no" para cumplir la misma función, pero las booleanas quedan más profesionales, además de ser las que menos memoria consumen. Se asignan igual que el resto: encendido = true, encendido = false.

4)Nil

El valor nil corresponde justamente a un valor nulo, es decir no es nada. Si se lo asignamos a una variable (chombres = nil, por ejemplo); la destruímos.

Alcance

a)Globales

Las variables además poseen alcance. Las variables como fueron declaradas previamente son globales, esto quiere decir que podrán ser usadas en cualquier parte del código, existirá en cualquier estructura que se use, y si se cambia su valor se cambiará en el código completo.
b)Locales
En cambio, si a la declaración de la variable le colocamos la palabra reservada **local**, la variable se creará sólo para esa estructura, es decir que si la declaramos dentro de una función, la variable existirá en esa función, pero si la llamamos fuera de la función, no existirá, y si existe es porque la creamos previamente, y seguramente tendrá otro valor, ya que aunque tengan el mismo nombre son dos variables distintas. Aún aunque sean locales, es recomendable no usar los mismos nombres en las funciones.

6.Funciones

Definiendo funciones

Una función es un bloque de código que al ser llamado, hace que el flujo del programa se detenga, se dirija a la función, ejecute sus sentencias, y regrese al flujo normal. Esto nos permite evitar las repeticiones, acortar el código, hacerlo más simple, más profesional, más ligero y más entendible.
Las funciones se definen con la palabra reservada Function, el nombre de la función, y sus argumentos entre paréntesis, separados por comas. El nombre de las funciones al igual que en el de las variables, debe ser descriptivo para ser entendible.
Los argumentos serán variables locales que ingrese el usuario al llamar la función, y que luego la función los maneje y los utilice para realizar una determinada orden.
Una vez escrito el conjunto de órdenes y sentencias, cerramos la función con **End**.

Return

Pero, antes de cerrar la función, podemos usar ***return*** *DEVOLUCIÓN*. Esto hace, que la función no se utilice para modificar algo, sino para obtener un dato que luego será manipulado. Si no usamos return, entonces la función será considerada como un procedimiento que hará cambios en el código. Por ejemplo, podemos hacer que dentro de una función se le cambie el valor a 5 variables, por lo que no es necesario usar return. Mientras que si una función se utilizara para que a una variable se le asigne el valor de x+y; se utilizaría return x+y, y al llamarla se pondría cuenta = hacercuenta(10,8)

Por ejemplo, supongamos que yo quiero hacer un programa que haga una determinada suseción de operaciones de unos gastos que voy a hacer, que sería muy repetitivo de no ser que no hiciera una función para llamarla cada vez que lo necesite. La función tomaría dos números, los sumaría entre sí, al resultado le sumaría 10 y luego lo multiplicaría por 3. Por lo tanto, mi programa sería:

```
--Declaramos la función, como "cuentas", con los argumentos a y b
Function cuentas(a,b)
  local primercuenta = a+b
  --Fíjense que utilizo una variable local, será sólo existente en ésta función
  local segundacuenta = (primercuenta+10)*3
  --Noten que utilizo paréntesis, sino multiplica 10*3 y luego se lo suma a primercuenta
  return segundacuenta
End
--Ahora llamo a la función, asignándole el valor de devolución (el return, osea
--segunda cuenta) a una variable que luego la imprimiré

vtotal = cuentas(7,8)
print ("El total a pagar será de "..vtotal.." pesos")
```

Ejercicios: *Probar llamar la función cuentas cambiando los argumentos y viendo que devuelve. *Lograr imprimir la función cuentas(a,b) directamente, sin asignársela a una variable previamente. *Probar llamar a la función en la línea de comandos directamente, sin ingresarla dentro de print, y ver que ocurre.

Funciones Predefinidas

Las funciones predefinidas son aquellas que ya vienen definidas por LUA, ya existen aunque no las veamos y las podemos utilizar sin necesidad de definirlas. Estas son una gran cantidad y se dividen según la función que cumplen, por ejemplo están las math que realizan cálculos matemáticos (funciones trigonométricas, raíz cuadrada, etc) os que devuelve datos del sistema (como la fecha, la hora, eliminar archivos, etc). Otras no pertenecen a ningún grupo, incluso print es una función predefinida, o acaso no notaron los paréntesis.
Una muy útil es dofile(ruta), que permite incluir un script o archivo de texto en nuestro programa. Es muy interesante para realizar bases de datos.
Las funciones tonumber(variable) y tostring(variable), como su nombre lo indica, permiten cambiar el tipo de la variable que le ingresemos.
Para más documentación sobre las funciones predefinidas (son bastantes pero no todas del todo necesario, por eso si realmente necesitan una búsquenla o pídanme ayuda), en Google, la mayoría está en inglés pero es entendible.

Ejercicio: Realizar un programa corto pero útil, que utilice variables y una (y si es posible varias) funciones, y por lo menos 2 funciones predefinidas; que esté bien ordenado, con nombres claros y

comentar todo lo posible. Puede estar basado en el ejemplo del colegio, de los pagos, o lo que quieran.

a)io.read()

Como les dije, hay muchas funciones predefinidas, todas muy útiles; pero dentro de ellas voy a resaltar algunas. Una de estas es io.read (similar a raw_input). Lo que hace es pedirle al usuario que ingrese algo y presione enter para continuar. Lo que escriba se le puede asignar, por ejemplo, a una variable...de esta manera:

[Ejemplo]
Código:

```
>nombre = io.read()
_                    El "_" representa el puntero que aparece para que el usuario
ingrese
algo, supongamos "Marta"
>print(nombre)
Marta
```

b)os.execute("pause")

Otra es os.execute("pause"), que es el equivalente a system("pause"). Lo que hace es pausar el programa y no continuar hasta que no se aprete una tecla, mostrando en pantalla el clásico mensaje "Presione una tecla para continuar..."

Parámetros extra y funciones vararg

Lua tiene una propiedad característica, en la que al llamar una función, si le colocamos parametros de más, no los tomará en cuenta. Esto se relaciona con las*funciones vararg*. Vararg proviene de "variable arguments" (probablemente). Dichas funciones, son iguales al resto sólo que en sus argumentos contiene una*expresión vararg*: **...** (tres puntos). Quiere decir que estas funciones no definen sus argumentos, sino que toman todos los extras y los administran para hacer ciertas devoluciones.

Les voy a explicar con ejemplos de funciones. Supongamos que nosotros tenemos la función sumar(a,b) (y que en el cuerpo sume a+b). Por lo que les dije primero, si nosotros llamamos sumar(1,4,6), va a sumar 1+4 (a+b) y devolverá 5, sin importar el 6. Si nosotros llamamos sumar(1), entonces devolverá error al intentar sumar 1 con nil (ya que al no valorar b, lo toma como nil).
Si en cambio definimos una función cuentas(a,b,...), estoy utilizando "...". Si lo llamamos como cuentas(1,5); tomará a y b como number, y a "..." lo descartará ya que no es necesario. Mientras que si llamamos como cuentas(1,5,6,7,8) tomará todos los numeros que ingresamos como number, gracias al vararg.

También podemos utilizar el múltiple return, colocándolos entre comas. Ejemplo:

```
function contar()
        return 1,2,3,4
end
```

Si esa función la usamos para llamar a cuentas: cuentas(contar()), sería como llamarla así: contar(1,2,3,4), por lo que tomará esos 4 valores como number.

7-Condicional

El condicional IF, es una estructura que permite colocar un cuerpo con un conjunto de órdenes que se ejecutarán sólo si la condición establecida es verdadera. Si conocen lo básico de cualquier lenguaje de programación actual sabrán muy bien de que hablo, sino, lo entenderán muy fácilmente.

IF, AND y OR

Es tan simple como esto: (Ejemplo) Si está lloviendo, entonces llevar un paraguas. La estructura es
If condición Then
Órdenes
End

Si nos olvidamos de alguna de éstas palabras (if, then, o end) o la colocamos en posición incorrecta, como en cualquier otra estructura obtendremos un error.

Además contamos con dos reservadas más, OR y AND. El and nos permitirá usar varias condiciones (Ejemplo en castellano: Si es día y es fin de semana entonces ir de picnic [IF es de dia AND es fin de semana THEN ir de picnic). En este caso se deben cumplir **las dos condiciones**, si se cumple una y la otra no entonces no es válido.
En cambio, si utilizamos OR, puede que se cumpla una o la otra, o ambas. (Si es fin de semana O es feriado, entonces no ir al colegio [IF es fin de semana OR es feriado THEN no ir al colegio])

Si el or se utiliza dentro de una condición sin iniciar una condición distinta, es decir en la misma condición, ejemplo: if calzado == "zapatilla" or "zapato" or "sandalia" then; entonces la condición va a observar la primer cadena, y si es correcta la va a tomar, sino va a proseguir con la siguiente y así hasta encontrar una correcta. Si no lo hace, entonces la condición es falsa.

Si la sentencia es una sola orden, no un conjunto, entonces se puede hacer en una sola linea IF condición THEN sentencia, sin necesidad de ponerlo en vertical.
Si la condición es una booleana, entonces no es necesario poner IF variable==TRUE THEN, se puede poner IF variable THEN, entendiéndose que es TRUE.

Signos de Comparación

Como les expliqué recién, IF se forman de IF condición THEN. Esa condición debe ser una expresión que solo permitirá ejecutar la sentencia si se cumple. Esta expresión se forma con caracteres especiales para crear una condición. Ellos son:

> (mayor que)
< (menor que)
== (igual a. RECUERDEN QUE el signo de comparación "igual que" es ==. Si colocamos un solo igual (=), estaremos utilizando el signo de asignación, para crear o asignar una variable)
>= (mayor o igual que. EL ORDEN DEBE SER > y luego =, Y SOLO EN ESE ORDEN)
<= (menor o igual que)
~= (desigual a)

Igualar a Nil: si igualamos a nil (==nil), entonces lo que haremos es preguntar si sierta variable existe o no.
if variable==nil (variable no existe)
if not variable==nil (variable existe)

if variable~=nil (variable existe)

NOT

Si a nuestro condicional le agregamos un Not detrás, entonces la condición debe ser incorrecta para que se pueda ejecutar la sentencia. Para que se entienda: Si no es fin de semana entonces ir al colegio (If not fin de semana then ir al colegio).
Con las booleanas, si colocamos IF not variable THEN, se entiende que queremos decir IF variable==false then. Cualquiera de las dos maneras es correcta, como les explicaba anteriormente.

Else y Else If

Else (que se puede traducir como "sino", "en cambio"), como la palabra lo dice, nos permite ejecutar un conjunto de órdenes en caso de que ocurra algo distinto a la condición sugerida. No es obligatorio, pero si muy útil. Su sintáxis es muy similar a muchos otros lenguajes de programación. Como vengo explicándoles "con lenguaje natural", voy a hacerlo con el Else para que lo comprendan bien.

SI está lloviendo ENTONCES (If...Then)
llevar un paraguas
SINO (Else)
dejarlo en casa

Pero, también, sino se cumple la condición principal, podemos establecer otra condición, si esa no se cumple podemos establecer otra, y así hasta que una se cumpla o ninguna lo haga, y siga de largo.
Para hacer esto podemos poner:
IF condición principal THEN
sentencia
ELSE IF otra condición THEN
sentencia
ELSE IF otra condición mas THEN
sentencia
ELSE
sentencia

Como verán termina, con "ELSE sentencia".
Ejercicio: Comprender que hace ese "ELSE sentencia" final.

Ahora, prosigan con los bucles. Allí podrán ver pequeños ejemplos de la utilización de IF y ELSE.

8-Bucles

Los bucles son estructuras que nos ahorrarán **muchas** iteraciones. Es necesario conocerlos muy bien para saber utilizarlos debido a que son muy recurrentes y el no conocerlos puede llevarlos a hacer códigos extremadamente raros, largos y poco vistosos, y aún menos entendibles.

For...do

Los bucles for son un bloque que al terminar de ejecutarse sus órdenes, en vez de continuar con el flujo del programa, volverá al inicio del bucle y repetirá las órdenes. Ésta repetición se realizará las veces que el usuario lo indique. La repetición puede ser un número directo o puede ser una variable,

por lo que, claramente, se pueden crear funciones donde el usuario ingrese el rango del bucle. Los for se usan de la siguiente forma: for variable = inicio,final do.
Vamos por partes, primero comenzamos por for(palabra reservada), luego seguimos por variable. ¿Qué es eso de variable? quiere decir que ingresaremos un nombre de variable (local) utilizada para el bucle que luego será destruída. Cada vez que el bucle se repita el valor de la variable aumentará en 1, es decir que si el rango del bucle es de 1 a 5, entonces el valor de la variable al principio será 1, luego 2, 3, 4 y por último 5.
inicio y final determinarán el rango, por ejemplo: 1,5 ; 1,10; 6,45; etc.
Finalmente do. Luego debajo de eso colocaremos el conjunto de órdenes a ejecutar, y debajo end para terminar.
Lo más interesante de esto, es ciertamente la variable. Ésta será la base del for, y la podemos manipular para incluirla en la sentencia a ejecutar. Por ejemplo, si yo quiero "contar ovejas", y no detenerme hasta no haber contado 12, podría hacer esto:

Código:
```
for numoveja = 1,12 do
--Vamos a hacerlo interesante, con if
if numoveja==1 then --Si el número es 1 entonces dice "Veo una oveja"
print ("Veo una oveja")
 else --En cambio, si son mas de 1 dice "Veo X ovejas", sino queda feo
print ("Veo "..numoveja.." ovejas")
end
end
print("No veo más ovejas :(")
```

Espero que ahora lo hayan entendido mejor.

Ejercicio: Realizar un for que cuente hasta 20, pero de 2 en 2 comenzando por 2 (imprimiendo con print:
2
4
6
8
10
etc...)

While...do

Bueno, este bucle repetirá un bloque mientras una expresión sea verdadera. Por ejemplo, si le decimos que repita las órdenes hasta que variable<=5 (variable sea menor o igual que 5), y dentro de las órdenes colocamos variable = variable+1 (siendo que variable empieza como = 1), el bloque se repetirá 5 veces, ya que irá sumando su valor hasta que en un momento variable será mayor que 5 y la expresión será falsa; en ese momento el bucle terminará y el programa seguirá su flujo normal.
En este caso, haremos una cuenta regresiva...

Código:
```
gregresiva = 10
while gregresiva>=0 do
If gregresiva>0 then
print (gregresiva)
 else
```

```
print ("Feliz Navidad!")
end
end
```
Devolución: 10
9
8
7
6
5
4
3
2
1
Feliz Navidad!

Ejercicio: Buscar sobre el bucle repeat...until

9-Arreglos y Tablas

Arreglos

Aveces, necesitamos una misma variable para varias cosas distintas. Por ejemplo, una variable que contenga el color de pelo de una persona podría ser colorpelo = rubio, pero si es entre 25 personas...una forma podría ser colorpelo1 = rubio, colorpelo2 = castaño, etc... y así con las 25; pero hay una forma de hacer esto más compacto y maleable. Los arreglos (array) son los indicados en este caso. Simplemente creamos una variable, y entre corchetes ([]) su índice. El índice indicara que número es. Por ejemplo, si queremos saber el color de pelo de Carlos, quien es el número 8 escribiremos: colorpelo[8], y nos devolvera el color que contenga. Para crear un arreglo, declaramos una variable sólo que su valor irá entre llaves, por ejemplo colorpelo = {}. Y a partir de ahí podemos crear más, colorpelo[1] = "colorado".
Por supuesto, podemos hacer una función simple para crear arreglos de forma rápida.
La función sería más o menos así:

Código:
```
function creararreglo(n) --para N colocar el tamaño del arreglo (con el ejemplo, 25)
           arreglo = {}
           for i = 1,n do
                       arreglo[i] = 0
           end
           return array
end

--llamando a la función
colorpelo = creararreglo(25)
```

Tablas

Pero hay veces, que necesitamos varias variables por persona. Por ejemplo, además del color de pelo, talvez necesitamos el color de ojos y la altura. Es lógico utilizar tres arreglos por persona, colordepelo[1] = "castaño", colordeojos[1] = "marrones", altura[1] = 180. Pero hay casos en los que se necesitarían muchos arreglos. Entonces las tablas nos pueden facilitar esto, sean 10 arreglos o 3 como el ejemplo que les di, es recomendable usar las tablas. Se crean de la misma forma (persona =

{}), sólo que dentro de las llaves ({}) irán directamente las variables que vamos a guardar, es decir: persona = {colordepelo = "",colordeojos = "",altura = 0}. Más adelante podemos modificar las variables que contenga la tabla. Para usarlos, como en los arreglos, colocaremos persona[INDICE].variable. Ej, para definir: persona[8].altura = 174. y para imprimir simplemente persona[8].altura.