Cadenas

Las cadenas son un tipo de expresión usadas para la definición de textos de longitud arbitraria. A través de las cadenas podremos definir todos los textos que sean necesarios en nuestra aplicación. El intérprete de Lua tiene un soporte limitado para la gestión de cadenas y se limita a la definición de cadenas y la concatenación de las mismas. A través de la librería `string`, tendremos acceso a funciones avanzadas de manipulación que veremos en este artículo.

Definición

Una cadena está formada por una sucesión de caracteres así como una serie de caracteres de control, como por ejemplo, retorno de carro. Una cadena en Lua se puede inicializar de tres formas distintas:

- Colocando una serie de caracteres entre comillas simples `'cadena'`.
- Colocando una serie de caracteres entre comillas dobles `"cadena"`.
- Colocando una serie de caracteres entre dobles corchetes `[[cadena]].`

Este ejemplo a continuación muestra los tres tipos de definición:

```
cadena1 = "Tutorial de Lua 1"
cadena2 = 'Tutorial de Lua 2'
cadena3 = [[Tutorial de Lua 3]]

print("La cadena 1 es: ", cadena1)
print("La cadena 2 es: ", cadena2)
print("La cadena 3 es: ", cadena3)
```

El resultado que obtendremos es el siguiente:

```
La cadena 1 es:         Tutorial de Lua 1
La cadena 2 es:         Tutorial de Lua 2
La cadena 3 es:         Tutorial de Lua 3
```

Caracteres de escape

Los caracteres de escape son secuencias de caracteres usadas en una cadena de caracteres y que permiten modificar la interpretación de éstos. Por ejemplo, en una cadena delimitada por comillas dobles, queremos usar comillas dobles dentro de la cadena, para ello las comillas dobles las representaremos usando `\"`: `"Mi texto \"entre comillas\""`. A continuación se muestra una lista de caracteres de escape.

Caracteres de escape	Uso	ASCII (decimal)
\n	New line	10
\r	Carriage return	13
\t	Tab	9
\v	Vertical tab	11
\\	Barra invertida	92
\"	Comillas dobles	34
\'	Comillas simples	39
\[	Corchete izquierdo	91
\]	Corchete derecho	93

Concatenación de cadenas con el operador `..`

El operador `..` permite la concatenación de cadenas en Lua, concatenando la expresión situada a la derecha del operador, a aquella que se encuentra a su izquierda.

```
> print("Tutoriales " .. "de " .. "Lua")
Tutoriales de Lua
> cadena = "Tutoriales " .. "de " .. "Lua"
> print(cadena)
Tutoriales de Lua
```

Manipulación de cadenas en Lua con la librería *string*

El intérprete de Lua provee un soporte muy limitado de la gestión de cadenas, proveyendo solamente las operaciones de definición y concatenación de cadenas. Para las operaciones más complejas, tales que, extracción de subcadenas, cálculo de longitud, conversiones, etc., Lua dispones de una una serie de funciones que permiten la manipulación de cadenas de una manera sencilla. Estas funciones están definidas en la librería `string`. A continuación veremos las diferentes funciones con más detalles así como algunos ejemplos de uso.

> La funciones de la librería string usan el índice 1. Esto significa que las funciones que devuelven el índice de un caracter, para el primer caracter, devuelven 1 (en otrols lenguajes devuelven 0).

Contenidos

string.find

`string.find(cadena, patrón, [,inicio [,plano)`

Busca la primera ocurrencia del `patrón` en la `cadena`. Si encuentra el patrón, devuelve un par de valores que contienen la posición de inicio y final. Si no encuentra nada devuelve `nil`. Por defecto la función `string.find` permite el uso de patrones de búsqueda, que son similares a las expresiones regulares, pero con limitaciones y una sintaxis diferente.

```
> string.find("Tutoriales de Lua en dbtutoriales.com", "Lua")
15      17
> string.find("Tutoriales de Lua en dbtutoriales.com", "Luna")
nil
```

El parámetro opcional `inicio`, un entero, permite definir la posición a partir de la cual se quiere realizar la búsqueda. Este parámetro puede ser positivo o negativo, para definir si se quiere empezar por la izquierda (positivo) o por la derecha (negativo).

```
> string.find("Tutoriales de Lua en dbtutoriales.com", "Lua", 5)
15      17
> string.find("Tutoriales de Lua en dbtutoriales.com", "Lua", 20)
nil
> string.find("Tutoriales de Lua en dbtutoriales.com", "Lua", -25)
15      17
```

El cuarto parámetro opcional `plano`, un booleano, permite desactivar el uso de patrones de búsqueda, de este modo los caracteres especiales pasados en `patrón`, son evaluados como caracteres a buscar y no como caracteres de patrón de búsqueda.

```
> string.find("Tutoriales de Lua 100% en dbtutoriales.com", "%sL")
14      15
> string.find("Tutoriales de Lua 100% en dbtutoriales.com", "%sL", 1, true)
nil
> string.find("Tutoriales de Lua 100% en dbtutoriales.com", "%", 1, true)
22      22
> string.find("Tutoriales de Lua 100% en dbtutoriales.com", "%")
stdin:1: malformed pattern (ends with '%')
stack traceback:
        [C]: in function 'string.find'
        stdin:1: in main chunk
        [C]: in ?
```

string.gsub

`string.gsub(cadena, patrón, reemplazo [, n)`

Devuelve una copia de `cadena` en la que todas las ocurrencias de `patrón` han sido reemplazadas por `reemplazo`. Si se especifica un número en el cuarto parámetro `n`, se reemplazarán las `n` ocurrencias de `patrón` en `cadena`. Esta función devuelve un segundo valor que representa el número de ocurrencias reemplazadas.

```
> string.gsub("manzana, piña, plátano", "piña", "pera")
manzana, pera, plátano   1
> string.gsub("manzana, piña, plátano, melón, piña", "piña", "pera")
manzana, pera, plátano, melón, pera     2
> string.gsub("manzana, piña, plátano, melón, piña", "piña", "pera", 1)
manzana, pera, plátano, melón, piña     1
```

string.format

`string.format(formato, arg1, arg2, ..., argn)`

Crea una cadena formateada a partir del `formato` y argumentos proporcionados. `formato` acepta un número arbitrario de marcadores de formato. El número de argumentos pasados debe coincidir con el número de marcadores pasados. Esta función es similar a la función `sprintf()` de C.

Los marcadores *s* y *q* aceptan valores *string*. Los marcadores *A, a, E, e, f, G* y *g* aceptan valores number. Los marcadores *c, d, i, o, u, X* y *x* aceptan valores integer.

```
> string.format("%c%c%c", 76, 117, 97)                -- Char (caracter)
Lua
> string.format("%e %E", 2.71828, 2.71828)            -- Exponent (exponente)
2.718280e+000 2.718280E+000
> string.format("%f", 2.71828)                        -- Float (coma flotante)
2.718280
> string.format("%g %g", 2.71828, 10e5)               -- Float o exponent (según tipo usa
uno u otro)
2.71828 1.000000e+006
> string.format("%d %i %u", -10, -10, -10)            -- Signed, signed, unsigned integer
(enteros con o sin signo)
-10 -10 18446744073709551606
> string.format("%s %s %q", "Tutoriales", "de", "Lua")  -- String, quoted (cadenas y entre
comillas)
Tutoriales de "Lua"
> string.format("%o %x %X", 255, 255, 255)            -- Octal, hexadecimal, hexadecimal
377 ff FF
> string.format("%a %A", 255, 255)                    -- Hexadecimal con exponente binario
(>= Lua 5.2)
0x1.fep+7 0X1.FEP+7
```

El marcador *q* pone la cadena pasada entre comillas usando caracteres de escape, de tal manera que la cadena devuelta puede ser evaluada de nuevo por el intérprete de Lua sin producir errores.

```
> string.format("%s %q %s", "Aprendiendo", "Lua", "con dbtutoriales.com")
Aprendiendo "Lua" con dbtutoriales.com
```

string.upper

`string.upper(cadena)`

Devuelve una copia de `cadena` en la que todas sus letras han sido convertidas a mayúsculas.

```
> string.upper("Tutoriales de Lua en dbtutoriales.com")
TUTORIALES DE LUA EN DBTUTORIALES.COM
```

string.lower

`string.lower(cadena)`

Devuelve una copia de `cadena` en la que todas sus letras han sido convertidas a minúsculas.

```
> string.lower("Tutoriales de Lua en dbtutoriales.com")
tutoriales de lua en dbtutoriales.com
```

string.reverse

`string.reverse(cadena)`

Devuelve una copia de `cadena` invertida

```
> string.reverse("Tutorial de Lua")
auL ed lairotuT
```

string.byte

`string.byte(cadena [, inicio [, fin)`

Devuelve el valor numérico de los caracteres comprendidos entre `inicio` y `fin` de `cadena`, ambos comprendidos, en la codificación ASCII. Si no se especifica `inicio`, se devuelve el valor del primer carácter. Se puede decir que es la función inversa de `string.char()`.

```
> string.byte("Lua")
76
> string.byte("Lua", 2)
117
> string.byte("Lenguaje Lua", 5, 10)           -- Devuelve los carácteres comprendidos entre
```

```
5 y 10 ambos incluidos, total 6 carácteres
117     97      106     101     32      76
> string.byte(" ")
32
```

string.char

`string.char(i1, i2, ... in)`

Devuelve una cadena representando los valores numéricos de carácter ASCII pasados como argumentos. Se puede decir que es la operación inversa de la función `string.byte()`;

```
> string.char(76, 117, 97)
Lua
```

string.len

`string.len(cadena)`

Devuelve la longitud de la `cadena`.

```
> string.len("Tutoriales de Lua")
17
```

string.rep

`string.rep(cadena, n)`

Devuelve una cadena en la que `cadena` aparece concatenado `n` veces.

```
> string.rep("Lua ", 3)
Lua Lua Lua
> string.rep("Lua\n", 3)
Lua
Lua
Lua
```

Revisión #11
Creado 20 junio 2023 13:57:13 por Guillermo
Actualizado 31 julio 2023 12:02:47 por Guillermo