String Library Tutorial

You can find details about the string library in section 5.3 of the Reference Manual [1]. For practical examples of usage of the string library, have a look at StringRecipes.

Note: In Lua string indices start at index value 1, not index value 0 (as they do in C).

string.byte(s [, i])

Return the numerical code the i-th character of the string passed.

```
> = string.byte("ABCDE")      -- no index, so the first character
65
> = string.byte("ABCDE",1)    -- indexes start at 1
65
> = string.byte("ABCDE",0)    -- we're not using C
> = string.byte("ABCDE",100)  -- index out of range, no value returned
```

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

Generate a string representing the character codes passed as arguments. Numerical codes are not necessarily portable across platforms.

```
> = string.char(65,66,67)
ABC
> = string.char()  -- empty string
```

string.dump(function)

Returns a binary representation of the given function, so that a later loadstring on that string returns a copy of the function. Function must be a Lua function without upvalues.

string.find(s, pattern [, init [, plain]])

Find the first occurance of the pattern in the string passed. If an instance of the pattern is found a pair of values representing the start and end of the string is

returned. If the pattern cannot be found `nil` is returned.

```
> = string.find("Hello Lua user", "Lua")
7       9
> = string.find("Hello Lua user", "banana")
nil
```

We can optionally specify where to start the search with a third argument. The argument may also be negative which means we count back from the end of the string and start the search.

```
> = string.find("Hello Lua user", "Lua", 1)  -- start at first character
7       9
> = string.find("Hello Lua user", "Lua", 8)  -- "Lua" not found again after character 8
nil
> = string.find("Hello Lua user", "e", -5)   -- first "e" 5 characters from the end
13      13
```

The pattern argument can be a regular expression which allows more complex searchs. See the PatternsTutorial for more information. We can turn off the regular expression feature by using the optional fourth argument `plain`. `plain` takes a boolean value and must be preceeded by `init`. E.g.,

```
> = string.find("Hello Lua user", "%su")          -- find a space character followed by "u"
10      11
> = string.find("Hello Lua user", "%su", 1, true) -- turn on plain searches, now not found
nil
```

string.format(formatstring, e1, e2, ...)

Create a formatted string from the format and arguments provided. This is similar to the `printf("format",...)` function in C. An additional option `%q` puts quotes around a string.

- c, d, E, e, f, g, G, i, o, u, X, and x all expect a number as argument.
- q and s expect a string.

```
> = string.format("%s %q", "Hello", "Lua user!")    -- string and quoted string
Hello "Lua user!"
> = string.format("%c%c%c", 76,117,97)              -- char
Lua
> = string.format("%e, %E", math.pi,math.pi)        -- exponent
3.141593e+000, 3.141593E+000
> = string.format("%f, %g", math.pi,math.pi)        -- float and compact float
3.141593, 3.14159
```

```
> = string.format("%d, %i, %u", -100,-100,-100)    -- signed, signed, unsigned integer
-100, -100, 4294967196
> = string.format("%o, %x, %X", -100,-100,-100)    -- octal, hex, hex
37777777634, ffffff9c, FFFFFF9C
```

string.gfind(s, pat)

This returns an pattern finding iterator. The iterator will search through the string passed looking for instances of the pattern you passed.

```
> for word in string.gfind("Hello Lua user", "%a+") do print(word) end
Hello
Lua
user
```

For more information on iterators read the ForTutorial and IteratorsTutorial. For more information on patterns read the PatternsTutorial.

string.gsub(s, pattern, replace [, n])

This is a very powerful function and can be used in multiple ways. Used simply it can replace all instances of the pattern provided with the replacement. A pair of values is returned, the modified string and the number of substitutions made. The optional fourth argument `n` can be used to limit the number of substitutions made:

```
> = string.gsub("Hello banana", "banana", "Lua user")
Hello Lua user  1
> = string.gsub("banana", "a", "A", 2)  -- limit substitutions made to 2
bAnAna  2
```

If a *capture* is used this can be referenced in the replacement string using the notation *%capture_index*, e.g.,

```
> = string.gsub("banana", "(an)", "%1-")    -- capture any occurances of "an" and replace
ban-an-a        2
> = string.gsub("banana", "a(n)", "a(%1)")  -- brackets around n's which follow a's
ba(n)a(n)a      2
> = string.gsub("banana", "(a)(n)", "%2%1") -- reverse any "an"s
bnanaa  2
```

If the replacement is a *function*, not a string, the arguments passed to the function are any captures that are made. If the function returns a string, the value returned is substituted back into the string. Just like `string.find()` we can use regular expressions to search in strings. Patterns are covered in the PatternsTutorial.

```
> = string.gsub("Hello Lua user", "(%w+)", print)  -- print any words found
Hello
Lua
user
        3
> = string.gsub("Hello Lua user", "(%w+)", function(w) return string.len(w) end) -- replace with lengths
5 3 4   3
> = string.gsub("banana", "(a)", string.upper)     -- make all "a"s found uppercase
bAnAnA  3
> = string.gsub("banana", "(a)(n)", function(a,b) return b..a end) -- reverse any "an"s
bnanaa  2
```

string.len(s)

Return the length of the string passed.

```
> = string.len("Lua")
3
> = string.len("")
0
> = string.len("Lua\000user")   -- Lua strings are 8 bit pure so \000 does not terminate
8
```

string.lower(s)

Make uppercase characters lower case.

```
> = string.lower("Hello, Lua user!")
hello, lua user!
```

string.rep(s, n)

Generate a string which is n copies of the string passed concatenated together.

```
> = string.rep("Lua ",5)
Lua Lua Lua Lua Lua
> = string.rep("Lua\n",3)
Lua
Lua
Lua
```

string.sub(s, i [, j])

Return a substring of the string passed. The substring starts at `i`. If the third argument `j` is not given, the substring will end at the end of the string. If the third argument is given, the substring ends at and includes `j`.

```
> = string.sub("Hello Lua user", 7)      -- from character 7 until the end
Lua user
> = string.sub("Hello Lua user", 7, 9)   -- from character 7 until and including 9
Lua
> = string.sub("Hello Lua user", -8)     -- 8 from the end until the end
Lua user
> = string.sub("Hello Lua user", -8, 9)  -- 8 from the end until 9 from the start
Lua
> = string.sub("Hello Lua user", -8, -6) -- 8 from the end until 6 from the end
Lua
```

string.upper(s)

Make all the lower case characters upper case.

```
> = string.upper("Hello, Lua user!")
HELLO, LUA USER!
```
