Strings Tutorial

Quotes

Strings are introduced in section 2.1 of the Reference Manual [1]. Strings can be defined using single quotes, double quotes, or double square brackets.

```
> = "hello"
hello
> = 'hello'
hello
> = [[hello]]
hello
```

Why so many ways to make a string? It allows you to enclose one type of quotes in the other. e.g.,

```
> = 'hello "Lua user"'
hello "Lua user"
> = "Its [[content]] hasn't got a substring."
Its [[content]] hasn't got a substring.
> = [[Let's have more "strings" please.]]
Let's have more "strings" please.
```

Double bracketed strings also have a few other special properties, discussed below.

Escape sequences

Lua can also handle C-like escape sequences. There are more details in the Reference Manual, section 2.1 [1].

```
> = "hello \"Lua user\""
hello "Lua user"
> = "hello\nNew line\tTab"
hello
New line        Tab
```

Escape sequences are not recognized when using double brackets, so:

```
> = [[hello\nNew line\tTab]]

hello\nNew line\tTab
```

Multiline quotes

Double square brackets can be used to enclose literal strings which traverse several lines. e.g.,

```
> = [[Multiple lines of text
>> can be enclosed in double square
>> brackets.]]
Multiple lines of text
can be enclosed in double square
brackets.
```

Nesting quotes

Only double square brackets allow nesting:

```
> = [[one [[two [[three]] ]] ]]
one [[two [[three]] ]]
```

Concatenation

Strings can be joined together using the concatenation operator " . . ". e.g.,

```
> = "hello" .. " Lua user"
hello Lua user
> who = "Lua user"
> = "hello "..who
hello Lua user
```

Numbers can be concatenated to strings. In this case they are *coerced* into strings and then concatenated. You can read more about coercion below.

```
> = "Green bottles: "..10
Green bottles: 10
> = type("Green bottles: "..10)
string
```

The string library

Lua supplies a range of useful functions for processing and manipulating strings in its standard library. More details are supplied in the StringLibraryTutorial. Below are a few examples of usage of the string library.

```
> = string.byte("ABCDE", 2) -- return the ASCII value of the second character
66
> = string.char(65,66,67,68,69) -- return a string constructed from ASCII values
ABCDE
> = string.find("hello Lua user", "Lua") -- find substring "Lua"
7       9
> = string.find("hello Lua user", "l+") -- find one or more occurances of "l"
3       4
> = string.format("%.7f", math.pi) -- format a number
3.1415927
> = string.format("%8s", "Lua") -- format a string
     Lua
```

Coercion

Lua performs automatic conversion of numbers to strings and vice versa where it is appropriate. This is called *coercion*.

```
> = "This is Lua version " .. 5.0 .. " we are using."
This is Lua version 5 we are using.
> = "Pi = " .. math.pi
Pi = 3.1415926535898
> = "Pi = " .. 3.1415927
Pi = 3.1415927
```

As shown above, during coercion, we do not have full control over the formatting of the conversion. To format the number as a string as we would like we can use the `string.format()` function. e.g.,

```
> = string.format("%.3f", 5.0)
5.000
> = "Lua version " .. string.format("%.1f", 5.0)
Lua version 5.0
```

This is explicit conversion using a function to convert the number, rather than coercion. You can read more about coercion of numbers in the NumbersTutorial.

FindPage · RecentChanges · preferences
edit · history
Last edited March 25, 2004 7:57 pm PDT (diff)