LUA - BASIC SYNTAX

Let us start creating our first Lua program!

First Lua Program

Interactive Mode Programming

Lua provides a mode called interactive mode. In this mode, you can type in instructions one after the other and get instant results. This can be invoked in the shell by using the lua -i or just the lua command. Once you type in this, press Enter and the interactive mode will be started as shown below.

```
$ lua -i
$ Lua 5.1.4  Copyright (C) 1994-2008 Lua.org, PUC-Rio
quit to end; cd, dir and edit also available
```

You can print something using the following statement −

```
print("test")
```

Once you press enter, you will get the following output −

```
test
```

Default Mode Programming

Invoking the interpreter with a Lua file name parameter begins execution of the file and continues until the script is finished. When the script is finished, the interpreter is no longer active.

Let us write a simple Lua program. All Lua files will have extension .lua. So put the following source code in a test.lua file.

```
print("test")
```

Assuming, lua environment is setup correctly, lets run the program using the following code −

```
$ lua test.lua
```

We will get the following output:

```
test
```

Let's try another way to execute a Lua program. Below is the modified test.lua file −

```
#!/usr/local/bin/lua

print("test")
```

Here, we have assumed that you have Lua interpreter available in your /usr/local/bin directory. The first line is ignored by the interpreter if it starts with # sign. Now, try to run this program as follows −

```
$ chmod a+rx test.lua
$./test.lua
```

We will get the following output.

```
test
```

Let us now see the basic structure of Lua program, so that it will be easy for you to understand the basic building blocks of the Lua programming language.

Tokens in Lua

A Lua program consists of various tokens and a token is either a keyword, an identifier, a constant, a string literal, or a symbol. For example, the following Lua statement consists of three tokens −

```
io.write("Hello world, from ",_VERSION,"!\n")
```

The individual tokens are −

```
io.write
(
"Hello world, from ",_VERSION,"!\n"
)
```

Comments

Comments are like helping text in your Lua program and they are ignored by the interpreter. They start with --[[and terminates with the characters --]] as shown below −

```
--[[ my first program in Lua --]]
```

Identifiers

A Lua identifier is a name used to identify a variable, function, or any other user-defined item. An identifier starts with a letter 'A to Z' or 'a to z' or an underscore '_' followed by zero or more letters, underscores, and digits $0 to 9$.

Lua does not allow punctuation characters such as @, $, and % within identifiers. Lua is a **case sensitive** programming language. Thus *Manpower* and *manpower* are two different identifiers in Lua. Here are some examples of the acceptable identifiers −

```
mohd         zara      abc     move_name    a_123
myname50     _temp     j       a23b9        retVal
```

Keywords

The following list shows few of the reserved words in Lua. These reserved words may not be used as constants or variables or any other identifier names.

and	break	do	else
elseif	end	false	for
function	if	in	local
nil	not	or	repeat
return	then	true	until
while			

Whitespace in Lua

A line containing only whitespace, possibly with a comment, is known as a blank line, and a Lua interpreter totally ignores it.

Whitespace is the term used in Lua to describe blanks, tabs, newline characters and comments. Whitespace separates one part of a statement from another and enables the interpreter to identify where one element in a statement, such as int ends, and the next element begins. Therefore, in the following statement −

```
local age
```

There must be at least one whitespace character *usuallyaspace* between local and age for the interpreter to be able to distinguish them. On the other hand, in the following statement −

```
fruit = apples + oranges   --get the total fruit
```

No whitespace characters are necessary between fruit and =, or between = and apples, although you are free to include some if you wish for readability purpose.

Loading [MathJax]/jax/output/HTML-CSS/fonts/TeX/fontdata.js