LUA - MODULES

What is a Module?

Module is like a library that can be loaded using *require* and has a single global name containing a table. This module can consist of a number of functions and variables. All these functions and variables are wrapped in to the table which acts as a namespace. Also a well behaved module has necessary provisions to return this table on require.

Specialty of Lua Modules

The usage of tables in modules helps us in numerous ways and enables us to manipulate the modules in the same way we manipulate any other Lua table. As a result of the ability to manipulate modules, it provides extra features for which other languages need special mechanisms. Due to this free mechanism of modules in Lua, a user can call the functions in Lua in multiple ways. A few of them are shown below.

```
-- Assuming we have a module printFormatter
-- Also printFormatter has a funtion simpleFormat(arg)
-- Method 1
require "printFormatter"
printFormatter.simpleFormat("test")

-- Method 2
local formatter = require "printFormatter"
formatter.simpleFormat("test")

-- Method 3
require "printFormatter"
local formatterFunction = printFormatter.simpleFormat
formatterFunction("test")
```

In the above sample code, you can see how flexible programming in Lua is, without any special additional code.

The require Function

Lua has provided a high level function called *require* to load all the necessary modules. It is kept as simple as possible to avoid having too much information on module to load it. The require function just assumes the modules as a chunk of code that defines some values which is actually functions or tables containing functions.

Example

Let us consider a simple example, where one function has the math functions. Let's call this module as mymath and filename being mymath.lua. The file content is as follows −

```
local mymath =  {}

function mymath.add(a,b)
   print(a+b)
end

function mymath.sub(a,b)
   print(a-b)
end

function mymath.mul(a,b)
   print(a*b)
end

function mymath.div(a,b)
```

```
   print(a/b)
end

return mymath
```

Now, in order to access this Lua module in another file, say, moduletutorial.lua, you need to use the following code segment.

```
mymathmodule = require("mymath")
mymathmodule.add(10,20)
mymathmodule.sub(30,20)
mymathmodule.mul(10,20)
mymathmodule.div(30,20)
```

In order to run this code, we need to place the two Lua files in the same directory or alternatively, you can place the module file in the package path and it needs additional setup. When we run the above program, we will get the following output.

```
30
10
200
1.5
```

Things to Remember

- Place both the modules and the file you run in the same directory.
- Module name and its file name should be the same.
- It is a best practice to return modules for require function and hence the module should be preferably implemented as shown above eventhough you can find other types of implementations elsewhere.

Old Way of Implementing Modules

Let me now rewrite the same example in the older way, which uses package.seeall type of implementation. This was used in Lua versions 5.1 and 5.0. The mymath module is shown below.

```
module("mymath", package.seeall)

function mymath.add(a,b)
   print(a+b)
end

function mymath.sub(a,b)
   print(a-b)
end

function mymath.mul(a,b)
   print(a*b)
end

function mymath.div(a,b)
   print(a/b)
end
```

The usage of modules in moduletutorial.lua is shown below.

```
require("mymath")
mymath.add(10,20)
mymath.sub(30,20)
mymath.mul(10,20)
mymath.div(30,20)
```

When we run the above, we will get the same output. But it is advised on to use the older version of

the code and it is assumed to less secure. Many SDKs that use Lua for programming like Corona SDK has deprecated the use of this.