LUA - STANDARD LIBRARIES

Lua standard libraries provide a rich set of functions that is implemented directly with the C API and is in-built with Lua programming language. These libraries provide services within the Lua programming language and also outside services like file and db operations.

These standard libraries built in official C API are provided as separate C modules. It includes the following −

- Basic library, which includes the coroutine sub-library
- Modules library
- String manipulation
- Table manipulation
- Math library
- File Input and output
- Operating system facilities
- Debug facilities

Basic Library

We have used the basic library throughout the tutorial under various topics. The following table provides links of related pages and lists the functions that are covered in various part of this Lua tutorial.

S.N.	Library / Method & Purpose
1.	**Error Handling** Includes error handling functions like assert, error as explained in Lua - Error Handling.
2.	**Memory Management** Includes the automatic memory management functions related to garbage collection as explained in Lua - Garbage Collection.
3.	**dofile [*filename*]** It opens the file and executes the contents of the file as a chunk. If no parameter is passed, then this function executes the contents of standard input. The errors will be propagated to the caller.
4.	**_G** Thus is the global variable that holds the global environment *thatis*,$_{G\cdot G} =_{G}$. Lua itself does not use this variable.
5.	

	getfenv [*f*] Returns the current environment in use by the function. f can be a Lua function or a number that specifies the function at that stack level − Level 1 is the function calling getfenv. If the given function is not a Lua function, or if f is 0, getfenv returns the global environment. The default for f is 1.
6.	**getmetatable *object*** If object does not have a metatable, returns nil. Otherwise, if the object's metatable has a "__metatable" field, returns the associated value. Otherwise, returns the metatable of the given object.
7.	**ipairs *t*** This functions fetches the indices and values of tables.
8.	**load *func*[, *chunkname*]** Loads a chunk using function func to get its pieces. Each call to func must return a string that concatenates with previous results.
9.	**loadfile [*filename*])** Similar to load, but gets the chunk from file filename or from the standard input, if no file name is given.
10.	**loadstring *string*[, *chunkname*]** Similar to load, but gets the chunk from the given string.
11.	**next *table*[, *index*]** Allows a program to traverse all fields of a table. Its first argument is a table and its second argument is an index in this table. next returns the next index of the table and its associated value.
12.	**pairs *t*** Suspends the running coroutine. The parameter passed to this method acts as additional return values to the resume function.
13.	**print . . .** Suspends the running coroutine. The parameter passed to this method acts as additional return values to the resume function.

14.	**rawequal** ***v1, v2*** Checks whether v1 is equal to v2, without invoking any metamethod. Returns a boolean.
15.	**rawget** ***table, index*** Gets the real value of table[index], without invoking any metamethod. table must be a table; index may be any value.
16.	**rawset** ***table, index, value*** Sets the real value of table[index] to value, without invoking any metamethod. table must be a table, index any value different from nil, and value any Lua value. This function returns table.
17.	**select** ***index, . . .*** If index is a number, returns all arguments after argument number index. Otherwise, index must be the string "#", and select returns the total number of extra arguments it received.
18.	**setfenv** ***f, table*** Sets the environment to be used by the given function. f can be a Lua function or a number that specifies the function at that stack level − Level 1 is the function calling setfenv. setfenv returns the given function. As a special case, when f is 0 setfenv changes the environment of the running thread. In this case, setfenv returns no values.
19.	**setmetatable** ***table, metatable*** Sets the metatable for the given table. *YoucannotchangethemetatableofothertypesfromLua, onlyfromC.* If metatable is nil, removes the metatable of the given table. If the original metatable has a "__metatable" field, raises an error. This function returns table.
20.	**tonumber** ***e*[*, base*]** Tries to convert its argument to a number. If the argument is already a number or a string convertible to a number, then tonumber returns this number; otherwise, it returns nil.
21.	**tostring** ***e*** Receives an argument of any type and converts it to a string in a reasonable format. For complete control of how numbers are converted, use string.format.
22.	

	type *v* Returns the type of its only argument, coded as a string. The possible results of this function are "nil" *astring*, *notthevaluenil*, "number", "string", "boolean", "table", "function", "thread", and "userdata".
23.	**unpack** *list*[, *i*[, *j*]] Returns the elements from the given table.
24.	**_VERSION** A global variable *notafunction* that holds a string containing the current interpreter version. The current contents of this variable is "Lua 5.1".
25.	**Coroutines** Includes the coroutine manipulation functions as explained in Lua - Coroutines.

Modules Library

The modules library provides the basic functions for loading modules in Lua. It exports one function directly in the global environment: require. Everything else is exported in a table package. The details about the modules library is explained in the earlier chapter Lua - Modules tutorial.

String manipulation

Lua provides a rich set of string manipulation functions. The earlier Lua - Strings tutorial covers this in detail.

Table manipulation

Lua depends on tables in almost every bit of its operations. The earlier Lua - Tables tutorial covers this in detail.

File Input and output

We often need data storage facility in programming and this is provided by standard library functions for file I/O in Lua. It is discussed in earlier Lua - File I/O tutorial.

Debug facilities

Lua provides a debug library which provides all the primitive functions for us to create our own debugger. It is discussed in earlier Lua - Debugging tutorial.

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