LUA - TABLES

Introduction

Tables are the only data structure available in Lua that helps us create different types like arrays and dictionaries. Lua uses associative arrays and which can be indexed with not only numbers but also with strings except nil. Tables have no fixed size and can grow based on our need.

Lua uses tables in all representations including representation of packages. When we access a method string.format, it means, we are accessing the format function available in the string package.

Representation and Usage

Tables are called objects and they are neither values nor variables. Lua uses a constructor expression {} to create an empty table. It is to be known that there is no fixed relationship between a variable that holds reference of table and the table itself.

```
--sample table initialization
mytable = {}

--simple table value assignment
mytable[1]= "Lua"

--removing reference
mytable = nil

-- lua garbage collection will take care of releasing memory
```

When we have a table **a** with set of elements and if we assign it to **b**, both **a** and **b** refer to the same memory. No separate memory is allocated separately for b. When a is set to nil, table will be still accessible to b. When there are no reference to a table, then garbage collection in Lua takes care of cleaning up process to make these unreferenced memory to be reused again.

An example is shown below for explaining the above mentioned features of tables.

```
-- Simple empty table
mytable = {}
print("Type of mytable is ",type(mytable))

mytable[1]= "Lua"
mytable["wow"] = "Tutorial"

print("mytable Element at index 1 is ", mytable[1])
print("mytable Element at index wow is ", mytable["wow"])

-- alternatetable and mytable refers to same table
alternatetable = mytable

print("alternatetable Element at index 1 is ", alternatetable[1])
print("mytable Element at index wow is ", alternatetable["wow"])

alternatetable["wow"] = "I changed it"

print("mytable Element at index wow is ", mytable["wow"])

-- only variable released and and not table
alternatetable = nil
print("alternatetable is ", alternatetable)

-- mytable is still accessible
print("mytable Element at index wow is ", mytable["wow"])
```

```
mytable = nil
print("mytable is ", mytable)
```

When we run the above program we will get the following output −

```
Type of mytable is  table
mytable Element at index 1 is  Lua
mytable Element at index wow is  Tutorial
alternatetable Element at index 1 is  Lua
mytable Element at index wow is  Tutorial
mytable Element at index wow is  I changed it
alternatetable is  nil
mytable Element at index wow is  I changed it
mytable is  nil
```

Table Manipulation

There are in built functions for table manipulation and they are listed in the following table.

S.N.	Method & Purpose
1	**table.concat** *table*[, *sep*[, *i*[, *j*]]] Concatenates the strings in the tables based on the parameters given. See example for detail.
2	**table.insert** *table*, [*pos*,]*value* Inserts a value into the table at specified position.
3	**table.maxn** *table* Returns the largest numeric index.
4	**table.remove** *table*[, *pos*] Removes the value from the table.
5	**table.sort** *table*[, *comp*] Sorts the table based on optional comparator argument.

Let us see some samples of the above functions.

Table Concatenation

We can use the concat function to concatenate two tables as shown below −

```
fruits = {"banana","orange","apple"}

-- returns concatenated string of table
print("Concatenated string ",table.concat(fruits))

--concatenate with a character
print("Concatenated string ",table.concat(fruits,", "))
```

```
--concatenate fruits based on index
print("Concatenated string ",table.concat(fruits,", ", 2,3))
```

When we run the above program we will get the following output –

```
Concatenated string  bananaorangeapple
Concatenated string  banana, orange, apple
Concatenated string  orange, apple
```

Insert and Remove

Insertion and removal of items in tables is most common in table manipulation. It is explained below.

```
fruits = {"banana","orange","apple"}

-- insert a fruit at the end
table.insert(fruits,"mango")
print("Fruit at index 4 is ",fruits[4])

--insert fruit at index 2
table.insert(fruits,2,"grapes")
print("Fruit at index 2 is ",fruits[2])

print("The maximum elements in table is",table.maxn(fruits))

print("The last element is",fruits[5])

table.remove(fruits)
print("The previous last element is",fruits[5])
```

When we run the above program, we will get the following output –

```
Fruit at index 4 is  mango
Fruit at index 2 is  grapes
The maximum elements in table is 5
The last element is mango
The previous last element is nil
```

Sorting Tables

We often require to sort a table in a particular order. The sort functions sort the elements in a table alphabetically. A sample for this is shown below.

```
fruits = {"banana","orange","apple","grapes"}

for k,v in ipairs(fruits) do
   print(k,v)
end

table.sort(fruits)
print("sorted table")

for k,v in ipairs(fruits) do
   print(k,v)
end
```

When we run the above program we will get the following output –

```
1 banana
2 orange
3 apple
4 grapes
sorted table
```

```
1 apple
2 banana
3 grapes
4 orange
```