41.2 LUA Grundlagen - Funktionen

Autor: Goetz
Quelle: Mein EEP-Forum

Im Grunde genommen sind Funktionen - Programmierer schauen jetzt bitte mal weg! - auch Variablen.

Jedenfalls gibt es da einige Gemeinsamkeiten.

Wie bei einer Variablen vergibt man auch bei einer Funktion den Namen selber.
Und es gelten die gleichen Regeln:

~ Am Anfang ein Buchstabe
~ Nur normale Buchstaben (also **keine Umlaute, kein ß etc**.)
~ Ziffern
~ und den Unterstrich
~ keine Sonderzeichen (also **kein % oder $ oder ! oder …**)
~ **kein Leerzeichen**

Eine weitere Gemeinsamkeit ist, dass in einer Funktion etwas gespeichert wird.
Allerdings mehr als nur eine Zahl.
Sondern zum Beispiel eine Rechenaufgabe.

Diese Aufgabe muss ich erst einmal entwerfen und so die Funktion festlegen.

Damit Lua versteht, dass ich die Aufgabe festlegen will, schreibe ich vor den Funktionsnamen das Wort **function** und hinter den Funktionsnamen die runden Klammern.

function ist einer dieser Lua Befehle, die ich zum Ende des vorigen Kapitels kurz angesprochen hatte.
function bedeutet:
Merke dir die folgende Aufgabe unter diesem Namen.

Also beispielsweise:
function Ausrechnen() 1 + 2
end

Weil in einer Funktion mehrere Aufgaben stecken können muss man Lua zeigen, wo die Aufgabenliste zu ende ist.
Dafür ist der Lua Befehl **end**.

Später kann ich dann die Funktion einfach mit ihrem Namen aufrufen.

Wenn ich also an späterer Stelle in mein Skript schreibe:
Ausrechnen()
dann rechnet Lua 1 + 2 aus.

Allerdings habe ich nichts davon, weil in der Funktion oben nicht drin steht, dass Lua sich das Ergebnis merken soll.

Würde ich stattdessen schreiben
function Ausrechnen()

Ergebnis = 1 + 2
end
dann hätte ich eine Variable, in der das Ergebnis von 1 + 2 gespeichert würde.

Allerdings würde der Wert nur so lange in dieser Variablen gespeichert, wie ich mich innerhalb der Funktion befinde. Sobald die Aufgaben in der Funktion erledigt sind, vergisst Lua diese Variable wieder. Der Grund dafür ist, dass man in großen Programmen eine unvorstellbare Anzahl an Variablen benötigt. Und weil in jeder Variablen etwas gespeichert wird, brauchen diese Platz. Davon möchte man nur so viel belegen, wie nötig. Deshalb ist es sinnvoll, manche Variablen zu vergessen, wenn sie ihren Zweck erfüllt haben.

Manche Variablen braucht man nur innerhalb einer Funktion.
Weil man sich nur kurz etwas merken muss.
Deshalb gilt für Variablen die Regel:
Wenn sie nur innerhalb der Funktion genannt werden, dann vergisst Lua sie, sobald die Funktion erledigt ist.

Variablen, die gleich zu Anfang des Skripts genannt werden - also noch vor allen Funktionsaufrufen - bleiben so lange erhalten, bis das Programm beendet wird.

In den paar Zeilen, aus denen jedes EEP Lua Skript zunächst besteht, steckt gleich vorne solch eine Variable.
Sie hat den Namen **I** und bekommt zunächst den Wert 0

I = 0

Weil diese Variable **I** schon vor der ersten Funktion genannt wird bleibt sie erhalten, bis das Lua Skript beendet wird.

Kurz darauf folgt die Festlegung einer Funktion. Sie bekommt den Namen **EEPMain**().
Main ist englisch und bedeutet in etwa Hauptsache

Mit
function EEPMain()
wird also das Hauptprogramm festgelegt.

Und das tut zunächst mal sehr wenig.
Aber auf eins, was das Hauptprogramm tut, möchte ich hier hinweisen:

Dort steht nämlich
I = **I** + 1
Und dieses **I** wurde etwas weiter oben als Variable festgelegt.

Eben war der Wert für **I** noch 0. Wenn die Hauptfunktion jetzt durchlaufen wird, dann ändert er sich zu 1
Beim nächsten Durchlauf wird daraus 2, dann 3, dann 4 und so weiter.

Wer jetzt kleinen Sinn darin erkennt, der hat schon viel verstanden.
Denn da ist auch keiner.
Mit diesem Wert, der in **I** gespeichert wird, passiert ansonsten nämlich nichts.

Löscht man oben die Zeile
I = 0
und in der Funktion die Zeile

I = **I** + 1
dann funktioniert das Skript genauso gut wie vorher.

Moooooment
Passiert mit diesem **I** wirklich gar nichts?
Doch!
Da steht nämlich noch die Zeile
print (**i**)

print ist ebenfalls ein Befehl
Eigentlich ist print das englische Wort für Drucken.
Aber seit Computer Bildschirme haben und nicht mehr alles auf lange Papierbänder ausdrucken müssen, steht dieser Befehl für „Schreib das in mein Bildschirmfenster“

Wenn man es ganz genau betrachtet, ist print kein Befehl, sondern eine Funktion, die schon fest definiert ist.
Das erkennt man an den Klammern.
In die Klammern schreibt man etwas, das die Funktion für ihre Aufgaben benutzen soll.
In diesem Fall einen Text
In den Klammern steht das vertraute **I**
Also eine Variable
Die Funktion **print**() schaut deshalb nach, was in der Variablen **I** gespeichert ist und schreibt es in das Ausgabefenster.

Wozu ist das gut?

Sehr einfach:
Lua gibt auf diese Weise im Ausgabefenster ein Lebenszeichen von sich.

Wenn man an den Zeilen, die von Anbeginn im Lua Skript stehen, nichts ändert und das Ausgabefenster aktiviert hat, dann kann man beobachten, dass in diesem Fenster fleißig gezählt wird.

Man sieht also, dass die Funktion EEPMain() immer wieder durchlaufen wird. Bei jedem Durchgang wird zu **I** Eins dazu gezählt und das Ergebnis ins Ausgabe Fenster geschrieben.

Alles, was man in das Lua Skript geschrieben hat, wird nur ausgeführt, so lange das Hauptprogramm läuft.
Wenn das stoppt, dann ist Lua beendet und hat keinen Einfluss mehr auf irgendetwas.

Fazit:
Ihr könnt die Festlegung der Variable
I = 1
und die Aufgabe
I = **I** +1
und die Aufgabe
print(**I**)
löschen, ohne das etwas schlimmes passiert.

Die Zählerei hört auf und das Ausgabefenster bleibt leer.
Die Funktion **EEPMain**() läuft trotzdem weiter und Lua arbeitet anstandslos.

Aber solch ein Lebenszeichen kann sehr nützlich sein.
Denn ohne diese Hilfe kann man nicht erkennen, ob Lua rennt

In der Funktion **EEPMain**() steht zum Schluss noch etwas. Nämlich
return 1

Das Wort **return** ist ebenfalls ein Befehl, der in Lua fest definiert ist.
In der englischen Sprache bedeutet return in etwa zurückgeben.

Das ist jedenfalls in Lua damit gemeint
return 1
heißt „Gib den Wert 1 zurück“

Am Anfang hatte ich beschrieben, wie man sich in einer Funktion das Ergebnis einer Aufgabe in einer Variable merken kann, um dieses Ergebnis später zu benutzen. Der Befehl **return** ist eine alternative Möglichkeit. Er macht es etwas bequemer das Ergebnis einer Funktion zu benutzen.

Ohne den Befehl **return** müsste ich folgendes tun, um das Ergebnis einer Aufgabe zu benutzen:

```
Ergebnis = 0

function Ausrechnen()
Ergebnis = 1 + 2
end


Ausrechnen()
print (Ergebnis)
```

Ich müsste also zuerst eine Variable festlegen,
dann die Funktion definieren
dann die Funktion aufrufen
und zuletzt das Ergebnis verwenden.

Mit **return** geht das mit etwas weniger Aufwand:

```
function Ausrechnen()
return 1 + 2
end

print (Ausrechnen())
```

Diesmal brauche ich keine zusätzliche Variable, sondern nur die Funktion.
In der Funktion steht, was sie zurück geben soll, wenn ich sie aufrufe.

Und wenn ich dann an anderer Stelle - im Beispiel mit dem Befehl **print**() - den Funktionsnamen benutze, dann passiert dort alles auf einmal: Die Funktion wird aufgerufen, erledigt ihre Aufgabe, gibt das Ergebnis zurück und setzt es dort ein, wo ich die Funktion aufgerufen habe.

Das ist eine ziemlich abstrakte Sache und gewiss nicht leicht nachvollziehbar.
Aber es ist ein Hinweis darauf, wofür der Befehl **return** gut ist.

Im speziellen Fall der Hauptfunktion **EEPMain**() wird **return** 1 benutzt, um die Funktion erneut aufzurufen, sobald sie durchlaufen wurde. Dass das passiert steht nirgendwo im Skript, sondern wurde sozusagen in die spezielle Funktion **EEPMain**() mit eingebaut.

Würde man die Zeile
return 1
löschen, dann liefe die Funktion **EEPMain**() nur ein einziges Mal und anschließend würde Lua sofort beendet.

Dieses **return** 1 ist also sozusagen die Vorbereitung für ein Notaus, welches man selber in das Lua Skript einbauen könnte.
Aber das müssen wir uns für später aufheben. Viel später!

Okay, wenn ich eine Funktion festlegen will, dann tu ich das mit dem Befehl **function**.
Aber was hat es mit den Klammern hinter den Funktionsnamen auf sich?

Die sind ein spezieller Speicherplatz, der für diese Funktion bereit gestellt wird.
Denn oft möchte man an eine Funktion etwas übergeben, das sie für ihre Aufgaben benutzen soll.

Ein prima Beispiel ist die Funktion **print** ()
Der möchte ich nämlich sagen, was sie drucken bzw. ausgeben soll.

Und das kommt in die Klammern.

Genauso kann ich es auch mit der Funktion **Ausrechnen**() tun, wenn ich sie entsprechend festlege:

```
function Ausrechnen(Beigabe)
Beigabe = Beigabe + 1
return Beigabe
end
```

Wenn ich jetzt an anderer Stelle schreibe
Ausrechnen(1)
dann wird die Funktion **Ausrechnen**() aufgerufen und die Variable **Beigabe** in der Funktion bekommt den Wert 1.
Die Funktion rechnet dann zu der Variablen Eins dazu, speichert das Ergebnis wieder in der Variablen **Beigabe** und gibt den gespeicherten Wert zurück.

Steht in meinem Programm an einer Stelle
print(**Ausrechnen**(1))
dann bekomme ich im Ausgabefenster die Zahl 2

Ich rufe nämlich die Funktion **print**() auf und sage ihr in den Klammern, dass sie die Funktion **Ausrechnen**() mit dem Wert 1 durchführen und das Ergebnis ausdrucken soll.

Ein ärgerlicher Fehler

Der User "SW-I" hat mich auf einen wirklich schlimmen Fehler aufmerksam gemacht

> Zitat von »SW-I«
> im Beitrag nutzt Du die Variable "I". Das machte mich erst etwas unsicher, weil es auf dem Rechner auch

so einen Strich gibt, der kein Buchstabe ist.

> Wichtiger: Du schreibst dann "Print (i)", also kleines "I".
> Falls das einen Fehler gibt, ist es evtl. besser, den Beitrag zu editieren (i --> I).

Ja, das gibt einen Fehler.
Denn ich schrieb ja im ersten Kapitel, dass jede andere Schreibweise eine andere Variable erzeugt.

john ist nicht das selbe wie **John**

Und das kleine **i** ist nicht das selbe wie das große **I**

Beim Buchstaben i sieht man das leider nur sehr schwer. Und ich hatte überlegt, ob ich für die Erklärung nicht besser einen anderen, lesbaren Buchstaben verwende.
Auch deshalb, weil das große i wie ein Strich aussieht und nicht wie ein Buchstabe.

Aber in den EEP Skripten wird nunmal das große I benutzt.
Und mir war wichtig, dass ihr das wiedererkennt. Dass ihr euch erinnert und denkt: "Ich weiß, warum das da steht. Das hat Götz mir erklärt."

In den Skripten wurde deshalb das I verwendet, weil es Programmier-Tradition ist.
Immer, wenn etwas gezählt werden soll, speichern Programmierer diese Zahl in einer Variablen i oder I.

Meist nehmen sie dafür das kleine i.

Und genau deshalb hat Trend das große I für diesen Zähler am Anfang benutzt.
Damit das kleine i für uns frei bleibt.

Im EEP Skript steht richtig
print(I)
also mit großem **I**

Stünde dort das kleine i, dann würde **print**() nicht die Zahl ausgeben, die im großen I gespeichert wurde.

Noch einmal:
Das i wird traditionell für Zähler verwendet.
Weil Programmierer gedanklich das Wort "integer" damit verbinden.
"integer" steht für ganzzahlig. Also nur ganze Zahlen. Keine Brüche.
Man zählt ja meistens 1 ... 2 ... 3 ...
Und nicht 1 ... 1,001 ... 1,002 ...

Prinzipiell kann man aber jeden Namen für diese Variable nutzen

Also beispielsweise auch das **A**

```
A = 0
function EEPMain()
A = A+1
print(A)
return 1
end
```

oder das Wort **Anzahl**

```
Anzahl = 0

function EEPMain()
Anzahl = Anzahl+1
print(Anzahl)
return 1
end
```

In allen Varianten passiert genau das gleiche.

Aber wenn ich schreibe

```
A = 0

function EEPMain()
A = A+1
print(a)
return 1
end
```

dann passiert etwas anderes. Weil ich **A** hoch zähle, aber **a** auf dem Bildschirm ausgebe.

Vielen Dank, SW-I, für diesen sehr wichtigen Hinweis!