Scripting Tutorial - Lesson 3

Download supporting files for this tutorial

Texas Instruments TI-Nspire Scripting Support Page

In this lesson we extend our use of TI-Nspire-based variables within our Lua script. In this way we can easily customize our document and, with a change to just one line, produce a page that can be very useful to those wanting to take advantage of Lua in their own TI-Nspire documents.

First, though, we will shift our tool use from Oclua to the TI-Nspire Scripting Tool. This is a Windows utility - Mac users need to find other alternatives. It is light and easy to use, but quite different than Oclua.

Lesson 3.1: Using the TI-Nspire Scripting Tool

While Oclua allows the entire scripting process to occur within a single application (TI-Nspire), three applications are required for use with the TI-Nspire Scripting Tool. Scripts are written within a word processing package, or just a simple text editor, such as Notepad++. The script is then saved with the ".lua" suffix ; it is thus saved as a Lua file.

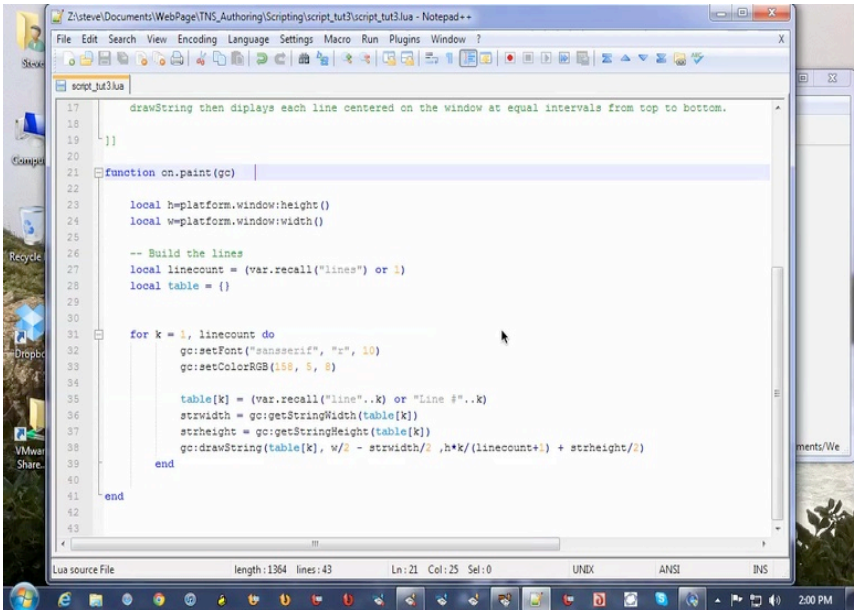
The TI-Nspire Scripting Tool then takes that file and stores the compiled version on the clipboard, from where it may be pasted into a blank TI-Nspire window.

1. Type or paste your script into Notepad ++ or a simple text editor.
2. Save as a .lua file.
3. Open TI-Nspire Scripting Tools.
4. Tools>Load Script App. Find the script and choose Open.
5. Though you will not see the script it will be â€œpastedâ€ into the Scripting Tool and a message will be displayed : Script App has been loaded to clipboard successfully. Choose OK.
6. Open a new window in TI-Nspire and paste the script. You will not see the script, but will see the result of the script.

This process is demonstrated in the short movie available here (again - do not hesitate

Click anywhere on this image for a video demonstration

to PAUSE!). The scripted window can be selected (ctrl-k) and copied (ctrl-c) and then pasted into new TI-Nspire documents if so desired. Unlike Oclua documents, closing and reopening the file once it has been created will not cause problems. The window remains live and active.



Lesson 3.2: Varying our Display Lines

One simple change in the script turns our short multi-line display document into a much more usable tool.

The line in which each line of the table is created in the previous script was:

table[k] = "Line #"..k

This glues the current value of "k" to the string "Line #", so that the first line will display as "Line #1", and so on.

You will remember that the **var.recall** command grabs the current value of a TI-Nspire variable and makes it available to Lua. In our example, we used this device to grab the number of lines we wished to display.

Now suppose we set up a variable called, for example, **line1**, the contents of which we want to be the first line? Then another called **line2** and so on? We could easily control the contents of our display, and change them at any time from within TI-Nspire.

For this to happen, we could simply change the line above to

table[k] = (var.recall("line"..k)

Click anywhere on this image for a video demonstration

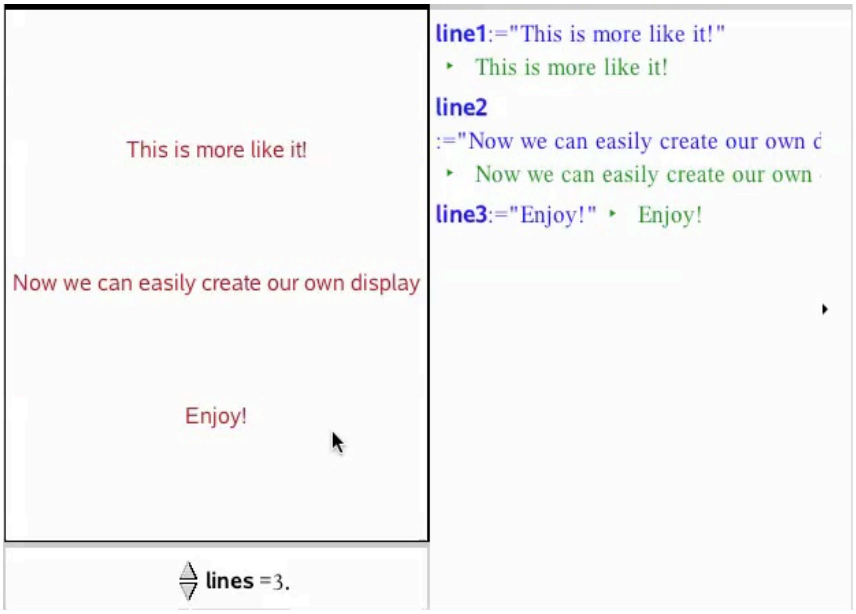
or "Line #"..k)

Study this new line and see if you can understand what is happening.

When k = 1, **var.recall** looks for a variable named **line1** and stores it as **table[1]** (the first entry of the table, "table"). If it cannot be found, then this line becomes just "line #1" (possibly a better choice might be a blank line, "".)

When k = 2, **table[2]** is defined as the current value of the variable **line2**, and so on.

So by storing values into these variables within your TI-Nspire document, they are immediately displayed within your Lua scripted page.



Lesson 3.3: Conditions

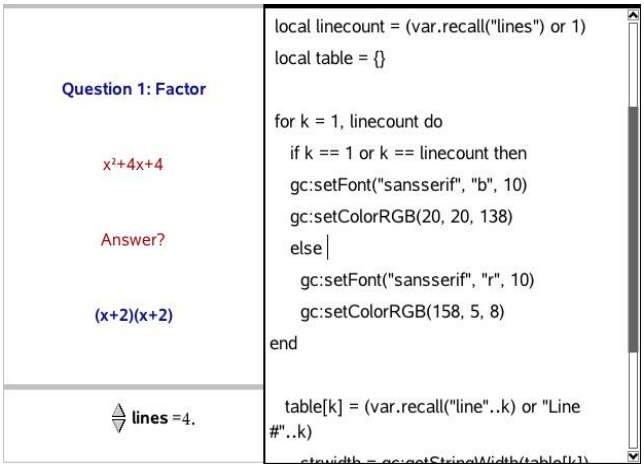
In our simple introduction to Lua scripting, we need one more ingredient: conditional statements: "if...then...else...end" (in TI-Nspire, the same syntax applies, except it finishes with "endif"!).

Suppose we would like our first and last lines to be highlighted in some way: perhaps a different color, or **bold** instead of plain style? This is easily achieved in our script.

```
for k = 1, linecount do

gc:setFont("sansserif", "r", 10)

gc:setColorRGB(158, 5, 8)

table[k] = (var.recall("line"..k) or "Line #"..k)

strwidth = gc:getStringWidth(table[k])

strheight = gc:getStringHeight(table[k])

gc:drawString(table[k], w/2 - strwidth/2 ,h*k/(linecount+1) +
strheight/2)

end
```

We just need to add a condition to the **setFont** and **setColor** commands! Some thing like

```
if k == 1 or k ==
linecount then
```



```
        gc:setFont("sansserif", "b", 10)
        gc:setColorRGB(20, 20, 138)
    else
        gc:setFont("sansserif", "r", 10)
        gc:setColorRGB(158, 5, 8)
    end
```

NOTE the use of the double equals sign to denote equality!!! A single equals sign denotes definition (k = 1). Watch this one!

Other than that, the rest should be pretty plain sailing.

And another lesson complete! Take some time to play and see what you can discover.

In our next lesson, we learn how to actually accept input directly into our Lua window, the basis for tutorial and quiz applications.