Scripting Tutorial - Lesson 1

Download supporting files for this tutorial

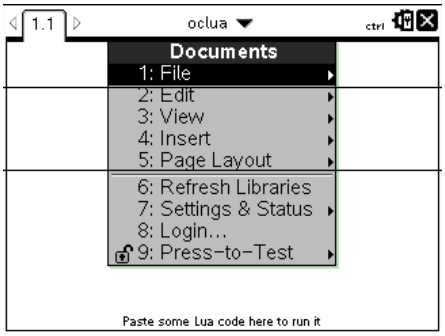
Texas Instruments TI-Nspire Scripting Support Page

TI-Nspire 3.0 adds powerful new functionality to the platform in the form of Lua scripting capabilities. Lua is a fast modern scripting language. Scripts written in Lua (even using a simple text editor) may be converted to TI-Nspire documents using the TI-Nspire Scripting Tool or by third-party tools. Such scripts add exciting new possibilities to TI-Nspire's native capabilities for creating documents for teaching and learning.

The power available to experienced programmers using the Lua scripting capabilities now built into TI-nspire is enormous, and the sky is the limit in terms of what is now possible, as exemplified by the range of sampler activities. However, there are opportunities for even those without a programming background to make use of these features in more limited but still potentially useful and powerful ways. These beginners' tutorials offer some practical ways in which teachers and students may begin to use Lua to create their own documents.

The focus here will be on using Lua initially to display text. Dynamic graphs and images will follow later.

Lesson 1.1: First Steps with Lua and TI-Nspire



For this first lesson, we will use the third-party Lua tool called Oclua, developed by Olivier Armand. Begin by downloading the Oclua package from the linked site or just grab the zipped folder of supporting documents available for this tutorial. Make frequent use of the **pause** button when watching videos in this series - it may take several views to follow everything that is happening.

Oclua supports actual Lua scripting within either TI-Nspire handheld or software on any platform. A script may be written in Notes, then copied and pasted into the first page of the document, oclua.tns, where it will be displayed as a Lua page. No need for additional software - fast and easy and a great way to get the feel for this new tool.

As you see from the animated gif image, once downloaded, open the file oclua.tns and you will see what looks like a blank page, except for the text at the bottom, which invites you to *"Paste some Lua code here to run it"* (Ignore the two horizontal lines you see on the image - they are an artifact of the software used to record this animation and will not appear in your version of this file).

Insert a new Notes page. As shown, type your first Lua function:

```
function on.paint(gc)
    gc:drawString("hello world", 0, 20)
end
```

The **indenting** of the second line is optional, but often helpful when working with longer scripts. The **case** of the text is NOT optional - Lua is a case-sensitive language and you must enter all commands exactly as shown here. So everything in this example is in lower case, except the "S" in "drawString".

As instructed in the image, select all (**ctrl-a** works just fine in Notes) and then go back to page 1.1 and paste (yes, you can use **ctrl-v**). You should see the text "hello world" appear in the top left corner of the window.

While most of the Lua scripts that you will see in the wild look enormously

complicated (and, in fact, they can be and usually are!) a successful Lua script can be as simple as this!

What do we learn from this?

- The syntax for defining a function in Lua is much the same as it is anywhere else:

```
function name(argument)
    some instruction
end
```

- In our example, the function is a standard Lua function, called "on.paint". You will see it in most (if not all) Lua scripts you look at, and it clearly delivers what it promises: "on" running this function, the contents are "painted" to the screen.
- "gc" stands for "graphics context" and is used when graphics of any type are defined. You see within the function, the single line of instruction is defined in terms of this "gc" term. Don't worry too much about this – you will get used to it very quickly.
- The instruction line of this function is also pretty explanatory: **gc:drawString("hello world", 0, 20)** means that, within the graphics context defined by this function, the string "hello world" will be drawn at location (0, 20) on the window.
- The coordinate system begins from the top left corner as the origin. So this text will begin at the left side of the window and 20 units down (yes, an x-y coordinate system just like we are used to, only the top of the screen is 0 and you count up to go down!)

*You should now take a few moments and **play**! Change the text, and change the x and y coordinates. Try to place the text in the center of the window – what does this teach you about the window dimensions? If you are on the software, switch between Computer and HandHeld views – you will see that, centering for one does NOT center for the other!*

Pause for breath before continuing.

But Why?

It is perhaps timely to stop for a moment and ask – *Why bother?* There are much easier ways to put text on a screen.

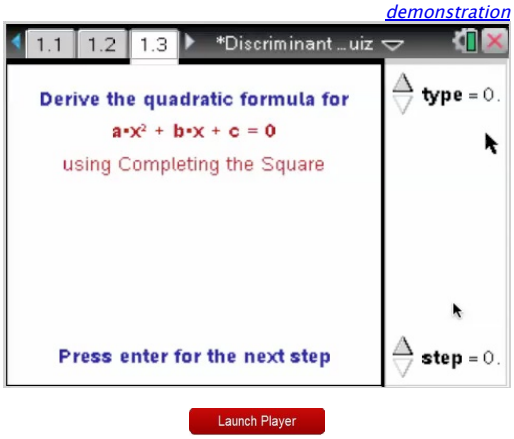
Two very important properties of a Lua window on TI-Nspire as opposed to say, the same thing displayed using Notes:

1. Click on any Lua-based window. Drag around – try and edit it. It is secure. Any text you display in such a window CANNOT be edited or changed by the user of your document.

   In Notes, of course, the user can readily change – and indeed break – whatever the author has taken the time and care to display there. AND, just as we can in Notes, the text we display can be static or dynamic. So we can easily create powerful learning tools just as we can with Notes, but in a secure display environment.

2. In the second part of this lesson, you will learn how to control the font size, color and style, and how to easily center text on a line or, indeed, in the center of the window. In Lua we have significant control over the way our text is displayed – much moreso than using the native TI-Nspire text facilities. In terms of color, you have ready access to millions of colors, and size is similarly unrestricted. Take a moment to watch the short

Click anywhere on this image to view a short video demonstration



to watch the short demonstration video to the right, "Derive the quadratic formula". It illustrates some of the possibilities for using Lua.

Lesson 1.2: A Little More Interesting?

So, back to work. Are you ready for a little more?

If you are not on page 1.2, return to that page. We will add a couple of lines to our on.paint script.

```
function on.paint(gc)

    gc:setFont("sansserif", "b", 12)

    gc:setColorRGB(158, 5, 8)

    gc:drawString("This is my text", 20, 20)

end
```

Click anywhere on this image for a video demonstration

Once again, study these new commands and see what they do - have a play and try different values. You will quickly appreciate that **setFont** takes three inputs (family ("serif" or "sanserif"), style ("b" for bold, "r" for regular, "i" for italic, or "bi" for bold italic) and, of course, size).

setColorRGB is a simple way to access lots of colors: RGB stands for Red, Green, Blue, where red is (255, 0, 0), Green is (0, 255, 0) and Blue (0, 0, 255). Clearly there are many colors in between and you can play around with numbers or, like me, you can go access lists online or just download the PDF that I use. My trusted color consultant advises me that burgundy (158, 5, 8) and navy (20, 20, 138) work very well together! And so much nicer than just a palette of 9 to 15 colors as currently available within TI-Nspire!

If you are still with us, then you are doing very well in your first steps with Lua. We will finish this first lesson by learning how to center that text, and in doing so introduce the important idea of local and global variables. In this case, we learn that Lua supplies all sort of useful commands, such as *platform.window:height()* and *platform.window:width()*. As the names suggest, these commands may be used to deliver to you the values for the current window. If we define these as variables, like w and h, then we can do some simple calculations to determine the best place to put our text! We will define these variables as local to our function, but they can be defined outside the function in a larger script. More on that later.

We introduce two more Lua commands as well: getStringWidth and getStringHeight. These should also be self-explanatory. Return to page 1.2 in Notes.

```
function on.paint(gc)

    local h=platform.window:height()

    local w=platform.window:width()

    gc:setFont("sansserif", "b", 12)

    gc:setColorRGB(158, 5, 8)

    local sw = gc:getStringWidth("This is my text")

    local sh = gc:getStringHeight("This is my text")

    gc:drawString("This is my text", w/2 - sw/2, h/2 + sh/2)

end
```

Click anywhere on this image for a video demonstration

So we define width and height as simple local variables, w and h. We can also get the string dimensions. Then to center the string horizontally, take the middle of the window (w/2) then go back by half the length of the string (w/2 - sw/2). Vertically the same, except that remember we count backwards in Lua - down is greater, up is less!

What Now?

That is all for lesson 1 - your turn to play!

Why not try to align your text to the right? Have a look at the centering that was just done and think about how that approach might be used for left and right alignment.

How about placing your text at the top of the page - or down the bottom?

Try some different colors - go wild!

In the next lesson, we learn how to make this text start to come alive, and how to use a table to organize multiple lines of text nicely laid out on your page!
