The main difference between TI-BASIC programming and Lua programming is the way the user interaction is turned inside out. BASIC programs are in control: they read input from users and display results whenever their logic dictates. Lua scripts, on the other hand, are reactive. They may only accept input in response to events such as key presses. And they only display results indirectly in response to the system requesting a repaint. It takes a new way of thinking to write programs in this fashion.

-John Powers 1/14/13

**Background**

‘Lua’ means ‘moon’ in Portuguese and the name derives from the language’s ancestry. It was developed at the Pontifical Catholic University of Rio de Janeiro in Brazil. Lua is an embedded language that uses an interpreter in the host operating system (in our case, the TI-Nspire OS). It is robust, fast, portable, embeddable, powerful, simple, small, and, best of all, free.

Writing a Lua program in TI-Nspire results in the creation of a custom application (app). This app can be controlled like any built-in app: it can be copied & pasted to other TI-Nspire documents, it can be split-screened and it can communicate with other apps via TI-Nspire variables. But the Lua program has to account for mouse actions and keystrokes in order to provide for interaction.

Use the Script Editor in the computer software (see Insert menus) to write your Lua programs. Lua apps are not created on the handheld although there are two special TI-Nspire documents that let you write and edit Lua programs (oclua.tns and TIDE.tns) for teaching purposes. And a special concern is that, when developing a Lua app, we have to plan for its behavior in the software, handheld and online Document Player.

Your first Lua app (*Lua is caseSensitive*):

![Script Editor](image)

**Notes:**

- `on.paint` is the event that’s responsible for displaying stuff.
- `gc = “graphics context”, the canvas` drawString to draw a string. There are other things to draw!

100, 100 are ‘screen coordinates’. 0, 0 is the upper left corner of the screen.

Click the ‘Set Script’ button on the toolbar to run the program. Use ‘Focus Script’ to being the app to the foreground. If there’s a syntax error in your program, the error message will appear at the bottom of the Editor in the ‘Console’. If you use `print( )` statements their output will also appear in the console.
**Necessary resources:**

- [www.lua.org](http://www.lua.org) – the home site of Lua

After reading *all* the documentation you will discover some useful functions:

```lua
platform.window:width()
platform.window:height()
```

... determine the width and height of the application (window) in which you are working.

So, to improve the program (*not very efficient, though*):

```lua
function on.paint(gc)
    W = platform.window:width()  -- width of screen
    H = platform.window:height()  -- height of screen
    str = "Hello, World!"  -- the string to draw
    sw = gc:getStringWidth(str)  -- string width
    sh = gc:getStringHeight(str)  -- string height
    gc:setColorRGB(128, 128, 128)  -- sets the paint color
    gc:drawString(str, (W-sw)/2,(H-sh)/2)  -- do the math!
end
```

**Images**

The *Script Editor* is used to convert picture files into strings that Lua uses to represent the image.

- **Insert Image** pastes a quoted string into the Editor at the current cursor position:

At the top of your code:

```lua
img = image.new(cursor_here_when_inserting_an_image)
i = image.copy(img, 50, 100)  -- make a copy that will fit
```

Then, in `on.paint(gc)` use:

```lua
gc:drawImage(i, 0, 0)  -- draw the copy
```

**Other Graphics**

- `gc:drawRect(x1, y1, w, h)`
- `gc:fillRect(x1, y1, w, h)`
- `gc:drawArc(cx, cy, r1, r2, a1, a2)`
  
  *etc... see the TI-Nspire documentation*
Events (on.things)
The Lua app responds to ‘events’ and the events are part of the on. module.

```lua
function init()  -- my own function
    x=0
end

function on.construction() -- when the app is constructed
    init()
end

function on.paint(gc) -- when the app needs to be repainted
    W=platform.window:width()
    H=platform.window:height()
    str="Hello, eWorld!"
    sw=gc:getStringWidth(str)
    sh= gc:getStringHeight(str)
    gc:drawString(str, (W-sw)/2,(H-sh)/2)
    gc:drawString(x,10,50)
end

function on.mouseMove() -- when the mouse is moved
    x=x+1
end

function on.mouseDown() -- when the left mouse button is pressed
    init()
+end
```

Classes
Classes provide a means of encapsulating objects with data and member functions similar to Java programming. Here’s a complete Lua example:

```lua
circle=class() -- the name of the class is circle

function circle:init(cx, cy, radius) -- called when a circle
    -- object is created
    self.x = cx
    self.y = cy
    self.r = radius
end

function circle:paint(gc) -- the routine to draw the circle
    gc:drawArc(self.x, self.y, self.r, self.r, 0, 360)
end

c = circle(50, 50, 50) -- c is now the circle object

function on.paint(gc)
    c:paint(gc)
end
```
Tables
Tables are used _extensively_ in Lua for organizing data. Similar to TI-Nspire lists but much more powerful. The _for_ loop is used to access the elements of a table:

```lua
x=65
myTable={3, 4, "John", x}

function on.paint(gc)
    for i = 1, #tabl do
        gc:drawString(myTable[i], 50, 20*i+20)
    end
end
```

Tables can be built ‘on the fly’ using:
```lua
table.insert(myTable,"thing")
```

For object-oriented programming, a table can contain _different_ objects.

**Communicating with other apps**
`var.monitor`, `var.store`, `var.recall` and `on.varChange` are used to pass values between the Lua app and the other apps in the problem.

Demo: make a Graphs or Geometry app and create a slider. Call the variable v. Create a Lua app and make the code:

```lua
function on.construction()
    var.monitor("v")    -- listen to the TI-Nspire variable v
    xpos = 0
end

function on.paint(gc)
    gc:drawArc(xpos, 100, 20, 20, 0, 360) -- draw a circle
end

function on.varChange()    -- when v changes
    xpos = var.recall("v") -- copy its value into the Lua app
end
```

You can combine the two apps onto one page by pressing ctrl-4 on the first page.

**Other resources:**
Steve Arnold’s Most Excellent Lua Tutorials:

An online Lua for TI-Nspire editor and emulator:

… works on tablets (including iPad) as well as computers. Write and test code here.