Scripting Tutorial - Lesson 17: Tips and Tricks I

Download supporting files for this tutorial

Download this page in PDF format

Texas Instruments TI-Nspire Scripting Support Page

math.eval

I have been having a lot of fun (and more than a little bit of frustration at times!) learning how to use this wonderful, powerful command.

After all, we are not like the rest of the Lua scripters out there, working on a blank canvass in a vacuum. We are working within a really powerful and full-featured mathematical environment - TI-Nspire. So while routines for things like finding the greatest common denominator (gcd) of two numbers are available out there on the Internet, should we not be able to reach across and use the GCD function that lives within TI-Nspire? This is where **math.eval** comes in.

Suppose we have two numbers, call them **num** and **den** and we wish to know if they have any common factors. If so we would like to divide these out and present the simplest form for the fraction num.."/"..den. For that we need the GCD of num and den.

NOTE that I am trying to show the Nspire stuff in red to try and expose what is going on here.

The simplest option would probably be to store **num** and **den** across to Nspire:

```
var.store("num", num)

var.store("den", den)
```

(No real problem with using the same name for the variables in both nspire and Lua since they will never actually, you know... meet.) Then...

local gcd = math.eval("gcd(num,den)")

Study this carefully to see what is happening here. So the argument of the math.eval function (in quotes) is what gets sent to Nspire, which understands that command and sends the answer back to Lua. Neat, eh?

A better option would probably be

local gcd = math.eval("gcd("..num..","..den..")")

since this does not involve sending any variables across the Lua/Nspire barrier. Again study and see how this is working.

Now stop and think about this and maybe see how potentially powerful this feature is. We can get TI-Nspire to compute any legitimate command that it understands.

Originally, I thought that this was limited to native TI-Nspire functions, but actually it works for user-defined functions as well. AND it works for CAS - with an important proviso: Lua can only deal with variable types that it recognises: numbers and strings, but NOT lists, expressions or matrices.

So suppose we wish to grab the result of a PolyRoots function applied to some function that we have defined: call it **fn**. For our purposes here, suppose **fn = "x^2-5"**. Then we could say

result = math.eval("polyroots("..fn..",x)")

Unfortunately, this would not work... but why not?

The result of polyroots is a list. We need to turn it into a string so that Lua can receive it. So a more successful approach would be

result = math.eval("string(polyroots("..fn..",x))")

Similarly if we were using CAS and we wanted to factor an expression - the result would be an algebraic expression and Lua cannot handle that. So we make sure that what is sent back to Lua is a string:

result = math.eval("string(Factor("..fn..",x))")

Finally, if we sent our function across to Nspire, it goes across as a string. So we would need to "unstring" it so that factor can work with it, then turn it back into a

string to send it back. Sorry, but this does make sense when you stick with it for a while. So an alternative approach would be...

var.store("fn", fn)

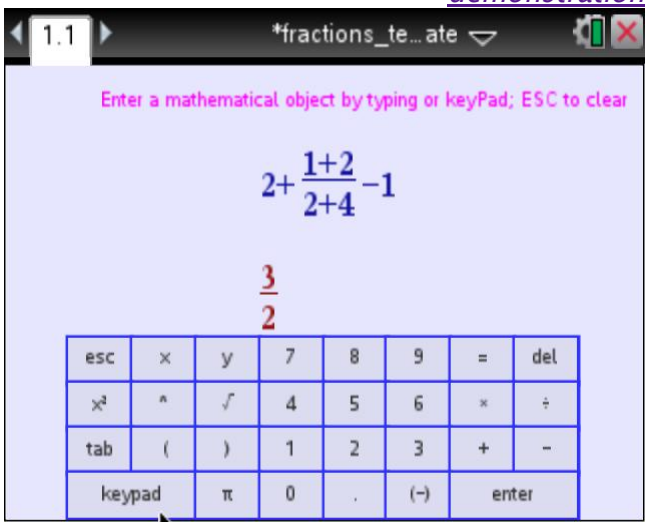
result = math.eval("string(Factor(expr(fn),x))")

I prefer the more elegant approach, avoiding sending variables across the divide if possible unless we really need them over there for something else. For example, suppose we also wanted to view the graph of our function **fn**? Then by storing it across in nspire, we could define **f1(x) = expr(fn)** and you will get the graph of whatever **fn** happens to be!

All of these various routines and ideas have been used to create the attached document, pretty.tns. In addition to the Pretty Print routine for algebraic expressions, the script includes a fraction display function, and math.eval routines that find roots for polynomials and evaluate numeric forms.

In the other more complete attached document, fractions_template.tns (shown here), there is even a keypad added.

Click anywhere on this image for a video demonstration



Back to Top

Home ← TI-Nspire Authoring ← TI-Nspire Scripting HQ ← **Scripting Tutorial - Lesson 17**
