PharmaSUG China 2018 - Hands-on Workshop

PROC LUA and why you should know it

Introduction

"Programming in SAS® has just been made easier" ... "Lua offers you a fresh way to write SAS programs" ...
... as Paul Tomas, Proc Lua developer at SAS Institute Inc., writes in his paper Driving SAS® with Lua (https://support.sas.com/resources/papers/proceedings15/SAS1561-2015.pdf) (SAS Global Forum, 2015)

"The Lua language seems likely to play an increasing role in the SAS world"
... as Amadeus Software, SAS Consultancy, UK, write here (https://amadeus.co.uk/sas-tips/using-lua-instead-of-sas-macro-language/).

"I believe that it will replace SAS Macro for big projects in the nearest future"
... Igor Khorlo, Founding Member, SAS User Group Germany (SUGG)

What is Lua?

- Lua is a "powerful, efficient, lightweight, embeddable scripting language" (as Lua describes itself)
- Lua was developed by Roberto Ierusalimschy (et al) at the Pontifical Catholic University of Rio de Janeiro in Brazil
- Lua is Portuguese for "moon" 月

Why Lua?

Paraphrased from A Comparison of the LUA Procedure and the SAS Macro Facility (Vijayaraghavan, 2017) (https://support.sas.com/resources/papers/proceedings17/SAS0212-2017.pdf):

- Need for an alternative to Macro language was felt by various solutions groups at SAS due to the inherent limitations of the latter
- Not feasible to enhance Macro language to the level of a modern scripting language like Lua
- Purpose of Lua is to script C-based software and SAS is written in C, so the two are a good fit for each other
- Lua provides superior debugging information

What else?

- PROC LUA provides an implementation of Lua 5.2 within Base SAS® (from 9.4)
- Low "entry requirements" for you as a SAS programmer:
  - User-friendly syntax, gentle learning curve
  - Direct access to the vast majority of SAS functions
  - PROC LUA is a brand new way to generate SAS code and it provides a realistic and powerful alternative to the SAS Macro language and `call execute()`
- Reputation for performance and memory efficiency
- Support for highly flexible data structures
- Support for OOP

Preliminary task 初步任务

Run the data step provided to create a DM data set in the WORK library.

First look at PROC LUA

```
-- Line comments in Lua start with two dashes
--[[
   And this is a
   block comment
]]
```

```
PROC LUA <restart|terminate>;
    SUBMIT;
       -- Lua code goes inside a SUBMIT block (but not always)
    ENDSUBMIT;
RUN;
```

Lua basics

First things first

- Lua is case sensitive
- Semicolons are optional and usually omitted

Data types

- Lua is dynamically typed
- Types are: number, string, boolean, table, nil, function, userdata and thread
- `nil` represents absence of a value, and is different from a SAS missing value
- Boolean values are `true` and `false`
  - **Only** `nil` and `false` evaluate to `false`
  - **Everything** else evaluates to `true` (including zero and SAS missing values!)

Naming rules

Same as SAS, except: can be as long as you like (subject to GPP)

Declaring variables

Variables have global scope within the current Lua state unless explicitly declared as local

```
local pi
pi = 3.1415926

local pi = 3.1415926
local v1, v2 = 'Hello', 'World'   -- Note list-style declaration and value assignme
nt
```

Writing to the log

```
print(v1..', '..v2..'!')          -- Writes "Hello, World!"
```

Exercise 1 练习 1

1. Use PROC LUA to write a message of your choice to the log. Declare and use one or more local variables for the message text.

Operators

Relational

< > <= >= == ~=

String

`..` (concatenation)
# (length of string or table e.g. #'Hello' returns 5)

Arithmetic

+ (addition)
– (subtraction, negation)
`*` (multiplication)
`/` (division)
`^` (exponentiation)
`%` (modulus)

Logical

`and` `or` `not`
The `and` operator returns its first argument if false, otherwise its second argument
The `or` operator returns its first argument if true, otherwise its second argument

SAS missing values

- Represented in PROC LUA by `sas.MISSING`
- Evaluate to `true` (we already knew that)

Convert to SAS missing value when `nil`

`<expression|variable> or sas.MISSING`

Basic control

if-then-else

```
if i == 1 then
   -- Conditional code
elseif i == 2 then
   -- More conditional code
else
   -- Yet more conditional code
end
```

Exercise 2 练习 2

1. Copy your code from Exercise 1 and add an "if-else" construct to it.

Loops

```
for i = 0, 15, 5 do        -- In SAS: do i = 0 to 15 by 5;
   print(i)
end

local i = 0
repeat                     -- In SAS: do ... until;
   print(i)
   i = i + 5
until (i > 15)

local i = 0                -- In SAS: do while ...;
while (i <= 15) do
   print(i)
   i = i + 5
end
```

Exercise 3 练习 3

1. Set up a basic "for-loop" using `i` as the iterator variable.
2. Write a message to the log using `i` within the message text.
3. Add the following piece of code AFTER the loop:

   ```
   print(i)
   ```

4. Does the `print(i)` output what you expect?

Tables

- Tables are fundamental to Lua
- Tables form the basis for highly flexible and customisable data structures, similar to lists in SCL
- Tables can contain items of different type (unlike SAS arrays)

Tables as arrays or lists

```
PROC LUA;
    SUBMIT;
        colours = {'red', 'blue', 'green', 'yellow', 999}
        print(colours[1], colours[5])
    ENDSUBMIT;
RUN;
```

Exercise 4 练习 4

1. Run the following code noting the syntax and output.

Using `ipairs()` to read through a list

```
PROC LUA;
    SUBMIT;
        -- local colours = {'red', 'blue', 'green', 'yellow'}
        for i, colour in ipairs(colours) do
            print(i, colour)
        end
    ENDSUBMIT;
RUN;
```

Exercise 5 练习 5

1. Run the following code noting the syntax and output.
2. How does Lua know about the *colours* table?

Lua tables as hash tables (key-value pairs)

```
PROC LUA restart;
    SUBMIT;
        local domain = 'SE'
        sp_domains = {CO='Comments',
                      DM='Demographics',
                      SE='Subject Elements',
                      SV='Subject Visits'}
        print(sp_domains.CO)
        print(sp_domains['DM'])
        print(sp_domains[domain])
        print(sp_domains['S'..'V'])
    ENDSUBMIT;
RUN;
```

Hash table values can be accessed via dot notation or [*key-as-literal-or-variable-or-expression*]

Exercise 6 练习 6

1. Run the following code noting the syntax and output.
2. Note the *restart* and associated log message.
3. Note the various ways of accessing the table values.

Using `pairs()` to read through a hash table

```
PROC LUA;
    SUBMIT;
        --[[
        local sp_domains = {CO='Comments',
                            DM='Demographics',
                            SE='Subject Elements',
                            SV='Subject Visits'}
        ]]
        for code, decode in pairs(sp_domains) do
            print(code, decode)
        end
    ENDSUBMIT;
RUN;
```

Exercise 7 练习 7

1. Run the following code noting the syntax and output.

Lua and SAS

Submitting SAS code

Ways to run SAS code

Two functions:

```
sas.submit   -- Submits immediately (restricted buffer size: ~32k)
sas.submit_  -- Delays submission until sas.submit() is encountered
```

Usage options:

```
1. sas.submit('SAS-code-as-string-literal')
2. sas.submit(SAS-code-as-string-variable)
3. sas.submit[[Raw-SAS-code]]
4. sas.submit(Variable-containing-[[Raw-SAS-code]])
```

Examples:

```
/** 1 **/
PROC LUA;
    SUBMIT;
        sas.submit('proc sort data=sashelp.class out=class; by name; run;')
    ENDSUBMIT;
RUN;

/** 2 **/
PROC LUA;
    SUBMIT;
        local mycode = 'proc sort data=sashelp.class out=class; by name; run;'
        sas.submit(mycode)
    ENDSUBMIT;
RUN;

/** 3 **/
PROC LUA;
    SUBMIT;
        sas.submit
        [[
          proc sort data=sashelp.class out=class;
              by name;
          run;
        ]]
    ENDSUBMIT;
RUN;

/** 4 **/
PROC LUA;
    SUBMIT;
        local mycode = [[
                          proc sort data=sashelp.class out=class;
                              by name;
                          run;
                        ]]
        sas.submit(mycode)
    ENDSUBMIT;
RUN;
```

Exercise 8 练习 8

1. Within Lua, subset the DM dataset you created in the Preliminary Exercise to contain screening failure subjects only.
2. Use `sas.submit` with square brackets as follows (note we will reuse the code in a later exercise):

```
PROC LUA;
     SUBMIT;
         sas.submit
         [[
           /* Your SAS code */
         ]]
     ENDSUBMIT;
 RUN;
```

3. Name the output data set SCREEN_FAILURES and keep the SUBJIDN variable only.
4. Ensure SCREEN_FAILURES is sorted by SUBJIDN. Hint: use `PROC SORT`

Substitution

`sas.submit` and `sas.submit_` take a table of key-value pairs as a second argument for resolution in the SAS code:

```
PROC LUA;
    SUBMIT;
        sas.submit('proc print data=sashelp.class; where age = @age@; run;', {age=1
2})
    ENDSUBMIT;
RUN;

PROC LUA;
    SUBMIT;
        local substit = {var='ag'..'e', val=10+3}
        local procprint = 'proc print data=sashelp.class; where @var@ = @val@; run;
'
        sas.submit(procprint, substit)
    ENDSUBMIT;
RUN;

PROC LUA;
    SUBMIT;
        -- Note requirement for normal function brackets when second argument is de
fined
        sas.submit
        ([[
          proc print data=@ds@;
              where age = 14;
          run;
        ]], {ds='sashelp.class'})
    ENDSUBMIT;
RUN;

PROC LUA;
    SUBMIT;
        local var, val = 'age', 15    -- Variables must be local for resolution
        local procprint = 'proc print data=sashelp.class; where @var@ = @val@; run;
'
        sas.submit(procprint)
    ENDSUBMIT;
RUN;
```

Exercise 9 练习 9

Amend your code from Exercise 8 to use substitution via a table to pass SUBJIDN into the SAS code.

Other ways to pass values into Lua I

Via the `SUBMIT` statement: `SUBMIT <"assignment(s);">;`

```
%let gr1 = Hello;
%let gr2 = how are you?;
PROC LUA;
    SUBMIT "greeting = '&gr1'";          /* greeting is global */
        print(greeting..', '.."&gr2")
    ENDSUBMIT;
RUN;
```

Exercise 10 练习 10

1. Run the above code and explain the output.

Using SAS functions in Lua

- The vast majority of SAS functions are available in Proc Lua, just add `sas.` to their function names like this: `sas.sum` `sas.scan` `sas.prxmatch`
- Functions which do not make sense outside of a data step are not available e.g. the `lag` function
- Note that with `sas.put` the format must be a string, variable or expression because Lua doesn't recognise a SAS format as a format
- Functions created by Proc FCMP are similarly called using the `sas.` prefix
- Proc Lua also provides a set of special use functions including dataset and table handling functions

```
PROC LUA;
    SUBMIT;
        local var = 'Hello Goodbye'
        print(sas.scan(var,1))
    ENDSUBMIT;
RUN;
```

Exercise 11 练习 11

1. Create some Lua variables and concatenate them using your favourite SAS `cat()` function.
2. Write the result to the log.

Other ways to pass values into Lua II

```
%let text = Hello, how are you?;
PROC LUA;
    SUBMIT;
        print(sas.symget('text'))
    ENDSUBMIT;
RUN;
```

Reading SAS datasets

Reading a SAS dataset I

- Within Lua, SAS data sets must be explicitly opened and closed
- Here we see the use of one of the data set handling functions provided in Proc Lua `sas.rows`
- Use to iterate over whole data set
- This approach is good for "narrow" tables with few columns because for each observation all variables are loaded into a Lua table (here called `obs`) whether subsequently needed or not

```
PROC LUA;
    SUBMIT;
        local dsid = sas.open('sashelp.class')
        for obs in sas.rows(dsid) do
            print(obs.name..' is '..obs['age']..' years old.')
        end
        sas.close(dsid)
    ENDSUBMIT;
RUN;
```

Exercise 12 练习 12

1. Rewrite the above code to loop through SASHELP.VTABLE and output to the log the name of each data set in the WORK library.
2. To do so, define a WHERE-clause to subset the data set to contain observations where LIBNAME="WORK" only

Reading a SAS dataset II

- Iterate over whole data set, load specific variables one observation at a time
- Two further dataset handling functions: `sas.next` and `sas.get_value`
- `sas.next` allows us to read through a SAS data set from start to finish
- `sas.get_value` provides access to specific variables, either by name or position
- Overcomes the performance disadvantage of reading entire observations into Lua

```
PROC LUA;
    SUBMIT;
        local dsid = sas.open('sashelp.class (where=(sex = "M"))')
        while sas.next(dsid) do
            print(sas.get_value(dsid,'name')..' is '..sas.get_value(dsid,'age')..'
years old.')
        end
        sas.close(dsid)
    ENDSUBMIT;
RUN;
```

Exercise 13 练习 13

1. Rewrite the above code to loop through SASHELP.VTABLE and output to the log the name of each data set in the WORK library
2. To do so, define a WHERE-clause to subset the data set to contain observations where LIBNAME="WORK" only
3. Use the macro variable INLIB within the WHERE-clause. Hint: use `sas.symget` and `sas.catx`

Exercise 14 练习 14

Using the code from Exercise 12:

1. Use `sas.submit` and `PROC PRINT` within the for-loop to output the data contained in the WORK data sets
2. Use substitution to pass the names of the data sets to the PRINT procedure
3. Remember that the variable obs is a table

Defining functions

- Functions are defined as follows:

```
<local> function function-name(arg1, arg2, arg3...)
    body of function
    return comma-separated-list-of-values
end
```

- Lua functions can return multiple values, including tables
- Here's a simple sort function which could be used in our data splitting exercise:

```
local function sortds(inds, by, outds)
    outds = outds or inds
    sas.submit [[
        proc sort data=@inds@ out=@outds@;
            by @by@;
        run;
    ]]
end
```

Checking for the existence of a data set

Exercise 15a 练习 15a

1. Run the following code. Is the output as expected?

```
PROC LUA;
    SUBMIT;
        local ds = 'SASHELP.IRIS'
        if sas.exist(ds) then
            print(ds..' exists!')
        else
            print(ds..' does not exist!')
        end
    ENDSUBMIT;
RUN;
```

Exercise 15b 练习 15b

1. Run the following code. Is the output as expected?

```
PROC LUA;
    SUBMIT;
        local ds = 'SASHELP.XXX'
        if sas.exist(ds) then
            print(ds..' exists!')
        else
            print(ds..' does not exist!')
        end
    ENDSUBMIT;
RUN;
```

Exercise 15c 练习 15c

1. Run the following code. Explain what is happening.

```
PROC LUA;
    SUBMIT;
        local ds = 'SASHELP.XXX'
        if sas.exists(ds) then
            print(ds..' exists!')
        else
            print(ds..' does not exist!')
        end
    ENDSUBMIT;
RUN;
```

Of course you can force Lua to return a Boolean from `sas.exist()` like this:

```
PROC LUA;
    SUBMIT;
        local ds = 'SASHELP.XXX'
        if sas.exist(ds) > 0 then
            print(ds..' exists!')
        else
            print(ds..' does not exist!')
        end
    ENDSUBMIT;
RUN;
```

Running PROC LUA inside SAS macros

- Embedding Lua code inside Proc Lua within a SAS macro is prohibited by the internal handling of SUBMIT/ENDSUBMIT blocks
- However, external Lua files CAN be referenced by PROC LUA via the `INFILE` option
- The external code referenced contains what *would* come within the SUBMIT/ENDSUBMIT block
- The `LuaPath` filename tells PROC LUA where to look for Lua files
- External Lua files have the extension `.lua` however the file is referenced without extension:

```
FILENAME LuaPath '/my/path/luafiles';
PROC LUA INFILE='myluafile';
RUN;
```

Some important aspects we haven't covered

- Loading data sets into tables
- Creating, writing and updating data sets
- File I/O

See the Tip Sheet for further info and try out at home!

Limitations of PROC LUA

- Lua code cannot be used directly inside a SAS macro
- Macros defined in PROC LUA are only available within the Lua state
- Not all Lua modules supported (e.g. `os`)
- Third-party Lua modules may not work in Proc Lua (particularly those not written for Lua 5.2)
- Lua's multi-threading capabilities are disabled in PROC LUA
- Current lack of familiarity with Lua in the pharma SAS community
- Prevailing conservatism within the pharma SAS community

Conclusion

- PROC LUA enables you to combine a powerful, modern scripting language with the functionality of SAS
- It's easy to get started with Lua - no steep learning curve to conquer before you can get going
- *"Programming in SAS® has just been made easier"*
- *"Lua offers you a fresh way to write SAS programs"*
- *"The Lua language seems likely to play an increasing role in the SAS world"*
- *"I believe that it will replace SAS Macro for big projects in the nearest future"*

Further reading

- https://www.lua.org (https://www.lua.org)
- https://en.wikibooks.org/wiki/Lua_Programming (https://en.wikibooks.org/wiki/Lua_Programming)
- https://www.tutorialspoint.com/lua/index.htm (https://www.tutorialspoint.com/lua/index.htm)
- SAS® Help Center: Concepts: PROC LUA (http://documentation.sas.com/?docsetId=proc&docsetTarget=p0t7zanjrat68hn1ue4mmy337rqe.htm&docsetVersion=9.4&locale=en)
- Execute Lua online (https://www.tutorialspoint.com/execute_lua_online.php)
- Driving SAS® with Lua (Tomas, 2015) (https://support.sas.com/resources/papers/proceedings15/SAS1561-2015.pdf)
- A Comparison of the LUA Procedure and the SAS® Macro Facility (Vijayaraghavan, 2017) (https://support.sas.com/resources/papers/proceedings17/SAS0212-2017.pdf)
- Using Lua instead of macro language (https://amadeus.co.uk/sas-tips/using-lua-instead-of-sas-macro-language/)
- Zerobrane Studio Lua IDE (https://studio.zerobrane.com/)
- Rowland's Lua area (https://github.com/rowland2425/Lua_PharmaSUG_China_2018)

Acknowledgments

Thanks go to my esteemed colleague Igor Khorlo who takes the blame for getting me interested in Lua in the first place, and who is a constant and ever willing source of good advice and valuable feedback.

Appendix

Exercise 1 练习 1

```
PROC LUA;
    SUBMIT;
        local t1, t2 = 'Hello Everybody!', 'Hope you enjoy the Lua HoW today!'
        print(t1..' '..t2)
    ENDSUBMIT;
RUN;
```

Exercise 2 练习 2

```
PROC LUA;
    SUBMIT;
        local t, hotcold = 30
        if t < 30 then
            hotcold = 'cold'
        else
            hotcold = 'hot'
        end
        print('The weather is '..hotcold..' in Beijing today!')
    ENDSUBMIT;
RUN;
```

Exercise 3 练习 3

```
PROC LUA;
    SUBMIT;
        for i = 1, 10 do
            print(i..' squared is '..i^2)
        end
    ENDSUBMIT;
RUN;
```

Exercise 8 练习 8

```
PROC LUA;
    SUBMIT;
        sas.submit[[
            proc sort data=DM out=SCREEN_FAILURES (keep=subjidn);
                where scrflny = 'Y';
                by subjidn;
            run;
        ]]
    ENDSUBMIT;
RUN;
```

Exercise 9 练习 9

```
PROC LUA;
    SUBMIT;
        sas.submit([[
            proc sort data=DM out=SCREEN_FAILURES (keep=@byvar@);
                where scrflny = 'Y';
                by @byvar@;
            run;
        ]], {byvar='subjidn'})
    ENDSUBMIT;
RUN;
```

Exercise 11 练习 11

```
PROC LUA;
    SUBMIT;
        local t1, t2 = 'I hope you are still', 'enjoying the Lua HoW!'
        print(sas.catx(' ', t1, t2))
    ENDSUBMIT;
RUN;
```

Exercise 12 练习 12

```
PROC LUA;
    SUBMIT;
        local dsid = sas.open('sashelp.vtable (where=(libname = "WORK"))')
        for obs in sas.rows(dsid) do
            print(obs.memname)
        end
        sas.close(dsid)
    ENDSUBMIT;
RUN;
```

Exercise 13 练习 13

```
%LET INLIB = WORK;
PROC LUA;
    SUBMIT;
        local dsid = sas.open(sas.cat('sashelp.vtable (where=(libname = "', sas.sym
get('inlib'),'"))'))
        while sas.next(dsid) do
            print(sas.get_value(dsid,'memname'))
        end
        sas.close(dsid)
    ENDSUBMIT;
RUN;
```

Exercise 14 练习 14

```
PROC LUA;
    SUBMIT;
        local dsid = sas.open('sashelp.vtable (where=(libname = "WORK"))')
        for obs in sas.rows(dsid) do
            sas.submit([[
                proc print data=@ds@;
                run;
            ]], {ds=obs.memname})
        end
        sas.close(dsid)
    ENDSUBMIT;
RUN;
```