SAS® 9.4 Proc Lua Tip Sheet

Basics

Proc Lua

```
PROC LUA <restart>;
   SUBMIT <'assignment(s);'>;
    -- Lua code
    -- Semicolons are optional and usually omitted
   ENDSUBMIT;
RUN;
PROC LUA terminate; /* terminates Lua state */
RUN;
filename LuaPath '<Path(s)(comma-delim) to Lua file(s)>';
PROC LUA INFILE='<Lua filename without extension>';
RUN;
```

Lua comments

```
-- This is a comment
--[[
   This is a block comment
]]
```

Main variable types

- *number, string, boolean, table*
- *boolean values are* `true` *and* `false`
  - *Only* `nil` *and* `false` *evaluate to* `false`
  - *Everything else incl.*`0` *(zero) evaluates to* `true`
- `nil` *represents absence of a value, and is different from a SAS missing value*
- `type()` *function returns the type of its argument*

Declaring variables

Variables have global scope unless explicitly declared as local

```
local phi = 1.618
local v1,v2 = 'Hello','World' -- list-style declaration
```

Writing to the log

```
print(v1..', '..v2..'!')  -- Writes "Hello, World!"
print [[Hello,
World!]]                  -- Multi line string
```

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`
- *Evaluates to true*

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
```

Loops

`for i = 0, 15, 5 do`
`   print(i)`
`end`
	`local i = 0`
	`while (i <= 15) do`
	`   print(i)`
	`   i = i + 5`
	`end`

`local i = 0`
`repeat`
`   print(i)`
`   i = i + 5`
`until (i > 15)`
	Output for each loop:
	`0`
	`5`
	`10`
	`15`

Use the `break` *statement to terminate a loop*	

Tables

Lua tables as arrays or lists

```
local colours = {'red', 'blue'}
print(colours[1])  -- Writes "red"
```

Using `ipairs()` to read through a list

```
for i, colour in ipairs(colours) do
   print(i, colour)
end
```

Lua tables as hash tables (name=value pairs)

```
local sp_domains = {CO='Comments',
                    DM='Demographics',
                    SE='Subject Elements',
                    SV='Subject Visits'}
```

```
print(sp_domains.CO)      -- Writes "Comments"
print(sp_domains['DM']) -- Writes "Demographics"
```

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

```
for code, decode in pairs(sp_domains) do
   print(code, decode)
end
```

Submitting SAS code

```
sas.submit('SAS-code-as-string-literal')
sas.submit(SAS-code-as-string-variable)
sas.submit([[Raw-SAS-code]],<table-with-substitution-values>)
sas.submit(Variable-containing-[[Raw-SAS-code]],
           <table-with-substitution-values>)
```

`sas.submit()` *immediately submits the sas code passed to the function*
`sas.submit_()` *delays submission until* `sas.submit()` *is encountered*

`sas.submit('proc print data=sashelp.class; run;')`

`local code = 'proc print data=sashelp.class; run;'`
`sas.submit(code)`

`sas.submit[[proc print data=sashelp.class; run;]]`

`local ds = 'sashelp.class'`
`sas.submit[[proc print data=@ds@;run;]]`

`sas.submit([[proc print data=@ds@;`
`          run;`
`        ]],{ds='sashelp.class'})`

`sas.submit_('proc print data=sashelp.class;')`
`sas.submit('run;')`

Dataset handling functions

`sas.open(dataset-name<,mode>)`
Opens a dataset, modes are i (read, the default), o (create), u (update), returns `nil` on failure, support for dataset options comes with SAS9.4M5
`sas.close(dsid)`
Closes a dataset opened with `sas.open()`
`sas.add_vars(dsid, variable-metadata-as-table)`
Adds one or more variables to a new dataset
`sas.attr(dsid,dataset-attribute-name)`
Returns an open dataset's specified attribute
`sas.exists(dataset-name)`
Returns `true` or `false` (different from `sas.exist` which returns 0 or 1, both of which evaluate to `true` in Lua)
`sas.new_table(dataset-name, table-of-variable-metadata-tables)`

Creates a new, empty dataset e.g:
`sas.new_table('work.birthdays',{`
`  {name='name',type='C',length=40,label='Name'},`
`  {name='date',type='N',length=8,label='Birthday',format='date9.'}`
`})`

Syneos. Health | Shortening the distance from lab to life.

SAS® 9.4 Proc Lua Tip Sheet

`sas.nobs(dsid)`	Returns the number of observations in an open dataset
`sas.nvars(dsid)`	Returns the number of variables in an open dataset
`sas.read_ds(dataset-name), sas.load_ds(dataset-name)`	Both return the contents of the dataset as a table, use only for small datasets, returns `nil` if the dataset does not exist
`sas.set_attr(dsid, attribute name, value)`	Sets the attribute of an open dataset
`sas.where(dsid, where-clause)`	Applies a where clause to an open dataset
`sas.write_ds(table, dataset-name)`	Creates a dataset from a table
`sas.append(dsid)`	Creates an empty observation and appends it to an open dataset
`sas.delobs(dsid)`	Deletes the current observation in an open dataset
`sas.get_value(dsid, variable-number|variable-name)`	Returns the value of a variable, specified by position or name, in the current observation
`sas.next(dsid)`	Moves to the next observation in an open dataset
`sas.put_value(dsid, variable-name, value)`	Populates the specified variable in an open dataset
`sas.rows(dsid)`	Iterates over observations in an open dataset and loads each row into a table
`sas.update(dsid)`	Updates an observation with values added by `sas.put_value()`
`sas.vars(dsid)`	Iterates over variables in an open dataset

Table handling functions

`table.concat(table-name<, "delimiter"><, start-position-as-integer<, end-position-as-integer>>)*`	Returns a string containing the contents of a table
`table.contains(table-name, value)`	Returns `true` if the specified table contains the value `value`
`table.insert(table-name, <position-as-integer, > value)*`	Inserts a value into a table, if no position is given the value is inserted at the end
`table.remove(table-name<, position-as-integer>)*`	Removes an entry in a table, if no position is given the last value is removed
`table.size(table-name)`	Returns the number of elements in a table
`table.sort(table-name<, comparison-function>)*`	Sorts table entries in ascending order, to sort in descending order specify a comparison function thus: `table.sort(t, function(a,b) return a>b end)`
`table.tostring(table-name)`	Returns a formatted string representation of the specified table.

** These functions were added in SAS9.4M5*

Reading a SAS dataset

Into a table (loads entire dataset into memory as table)

```
class = sas.load_ds('sashelp.class')
print(table.tostring(class)) -- Writes table
print(class[1]['name'])      -- Writes value
```

Row by row (loads entire observation into table on each iteration)

```
local dsid = sas.open('sashelp.class')
for obs in sas.rows(dsid) do
   print(obs.name)
end
sas.close(dsid)
```

Row by row (loads selected variables only on each iteration)

```
local dsid = sas.open('sashelp.class')
while sas.next(dsid) do
   print(sas.get(dsid,'name'))
end
sas.close(dsid)
```

Writing a SAS dataset

Example

```
-- Define new dataset
sas.new_table('squares', {
    {name='n',  type='n', length=8, label='N'},
    {name='n2', type='n', length=8, label='N squared'},
})
-- Open dataset in update mode
local dsid = sas.open('squares', 'u')
for n = 1, 10 do
    -- Append new but empty observation
    sas.append(dsid)
    -- Populate variables
    sas.put_value(dsid, 'n', n)
    sas.put_value(dsid, 'n2', n^2)
    -- Commit observation to dataset
    sas.update(dsid)
end
-- Close dataset
sas.close(dsid)
```

Update a SAS dataset

- *As above, without definition of new dataset*
- *Use WHERE-clause to isolate rows to update*

Functions

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

Examples

```
local function sortds(inds, by, outds)
    outds = outds or inds
    sas.submit [[
        proc sort data=@inds@ out=@outds@;
            by @by@;
        run;
    ]]
end
local function dateparts(sasdate)
    return sas.put(sas.day(sasdate),'z2'),
           sas.put(sas.month(sasdate),'z2'),
           sas.year(sasdate)
end
local d, m, y = dateparts(sas.today())
```

Access to SAS functions

Prefix (most) SAS functions with `sas.` *e.g:* `sas.date()` `sas.substr()` `sas.prxmatch()` `sas.symget()` `sas.catx()` *etc.*

Note: no native support for regular expressions in Lua, use `sas.prx*()` *functions*

File I/O

```
local p = 'c:\\temp\\luaiotest.txt'
-- Create new file (or overwrite an existing one)
f = io.open(p, 'w')
-- Set default output file to luaiotest.txt
io.output(f)
-- Write text to file
io.write('The quick brown fox\njumps over the lazy dog.')
-- Close file
f:close()
-- Open file for reading
f = io.open(p, 'r')
-- Set default input file to opened file
io.input(f)
-- Read file using io.lines
for t in io.lines() do
    print(t)
end
-- Close file
f:close()
```

Further reading via:

https://github.com/rowland2425/Lua_PharmaSUG_China_2018