penlightplus

Additions to the Penlight Lua Libraries

Kale Ewasiuk (kalekje@gmail.com)

2025-02-14

Package Options and Set-Up

This package first loads the LaTeX `penlight[import]` package:
`https://ctan.org/pkg/penlight?lang=en`.
Documentation for the Lua penlight package can be found here:
`https://lunarmodules.github.io/Penlight/index.html`.
The `pl` option may be passed to this package to create an alias for `penlight`.
A portion of this package to facilitate the creation, modification, and usage of the Lua table data structure through a LaTeX interface has been moved to a separate package called `luatbls`:
`https://ctan.org/pkg/luatbls`.

The following global Lua variables are defined:

`__SKIP_TEX__` If using the `penlightplus` package with `texlua` (good for troubleshooting), set this global before loading `penlight`
`__PL_GLOBALS__` If using this package with `texlua` and you want to set some functions as globals (described in next sections), set this variable to `true` before loading `penlight`
`__PL_NO_HYPERREF__` a flag used to change the behaviour of some functions, depending on if you don't use the hyperref package
`__PDFmetadata__` a table used to store PDF meta-data for pdfx package.

globals option

Since this package uses the penlight `import` option, all `stringx` functions are injected into the `string` meta-table and you can use them like so: `'first name':upfirst()`. But if the package option `globals` is used, many additional globals are set for easier scripting. `pl.hasval`, `pl.COMP`, `pl.utils.kpairs`, `pl.utils.npairs` become globals. `pl.tablex` is aliased as `tbx` (which also includes all native Lua table functions), and `pl.array2d` is aliased as `a2d`.

texlua usage

If you want to use `penlightplus.lua` with the `texlua` interpreter (no document is made, but useful for testing your Lua code), you can access it by setting `__SKIP_TEX__ = true` before loading. For example:

```
package.path = package.path .. ';'..'path/to/texmf/tex/lualatex/penlightplus/?.lua'
package.path = package.path .. ';'..'path/to/texmf/tex/lualatex/penlight/?.lua'
penlight = require'penlight'

__SKIP_TEX__ = true  --only required if you want to use
                     --penlightplus without a LaTeX run
__PL_GLOBALS__ = true -- optional, include global definitions

require'penlightplus'
```

penlight additions

Some functionality is added to penlight and Lua.

General Additions

`pl.hasval(x)` Python-like boolean testing
`COMP'xyz'()` Python-like comprehensions:
`https://lunarmodules.github.io/Penlight/libraries/pl.comprehension.html`

`_Gdot(s)` return a global (may contain dots) from string

`clone_function(f)` returns a cloned function
`operator.strgt(a,b)` compares strings a greater than b (useful for sorting)
`operator.strlt(a,b)` compares strings a less than b (useful for sorting)

`math.mod(n,d)` math modulus
`math.mod2(n)` mod with base 2

`pl.utils.filterfiles(dir,filt,rec)` Get files from dir and apply glob-like filters. Set rec to `true` to include sub directories

`pl.trysplitcomma(s)` will try to split a string on comma (and strip), but if is a table, leave it

`pl.findfiles{}` or `findfiles'kv'` is an updated version of `filterfiles`. Pass a table or a luakeys kv string as the only argument. Valid table options are: `fn, dir, ext, sub`.

`pl.char(n)` return letter corresponding to 1=a, 2=b, etc.
`pl.Char(n)` return letter corresponding to 1=A, 2=B, etc.

string additions

`string.upfirst(s)` uppercase first letter
`string.delspace(s)` delete all spaces
`string.trimfl(s)`remove first and last chars
`string.splitstrip(s, sp, st)` split by sp (default comma) followed by strip (default whitespace)
`string.split2(s, sep1, sep2, st)` split a string twice (creates a 2d array), first by sep1 (default comma), then by sep2 (default =), with option to strip (default true)
`string.appif(s, append, bool, alternate)`
`string.gfirst(s, t)`return first matched patter from an array of patterns t
`string.gextract(s,pat)` extract a pattern from a string (returns capture and new string with capture removed)
`string.gextrct(s,pat,num,join)` extract a pattern from a string (returns capture and new string with capture removed), can specify a number of extractions. if join is specified, captures will be joined, otherwise a list is returned
`string.totable(s)` string a table of characters
`string.tolist(s)` string a table of characters
`string.containsany(s,t)` checks if any of the array of strings `t` are in `s` using `string.find`
`string.containsanycase(s,t)` case-insensitive version
`string.delspace(s)` clear spaces from string
`string.subpar(s, c)` replaces `\\par` with a character of your choice default is space
`string.istexdim(s)` checks if a string is a valid tex dimension (eg. mm, pt, sp)
`string.fmt(s, t, fmt)` format a string like `format_operator`, but with a few improvements. `t` can be an array (reference items like `\$1` in the string), and `fmt` can be a table of formats (keys correspond to those in `t`), or a string that is processed by luakeys.
`string.parsekv(s, opts)` parse a string using a `luakeys` instance (`penlight.luakeys`). A kv-string or table can be used for opts.
`string.hasnoalpha(s)` string has no letters
`string.hasnonum(s)` string has no numbers
`string.isvarlike(s)` string is 'variable-like', starts with a letter or underscore and then is alphanumeric or has underscores after

tablex additions

`tablex.fmt(t, f)` format a table with table or key-value string f
`tablex.list2comma(t)` Use oxford comma type listing, e.g. A, B, and C
`tablex.strinds(t)` convert integer indexes to string indices (1 -> '1')
`tablex.filterstr(t,e,case)` keep only values in table t that contain expression e, case insensitive by default.
`tablex.mapslice(f,t,i1,i2)` map a function to elements between i1 and i2
`tablex.listcontains(t,v)` checks if a value is in a array-style list
`tablex.kkeys(t)` returns keys that are non-numeric (like kpairs)
`tablex.train(t,seq,reind)` return a sable based on `pl.seq.tbltrain`, `reind` will make numerical keys ordered from 1

List additions

`List:inject(l2, pos)` injects a list (l2) into a list at position. Set pos=0 to inject at end.

seq additions

A syntax to produce sequences or a 'train' of numbers is provided. This may be useful for including pages from a pdf, or selecting rows of a table with a concise syntax.

`seq.prod(t1, t2)` iterate over the cartesian product of t1 and t2
`seq.train(trn, len)` produces a pl.List according to the arguments
`seq.itrain(trn, len)` produces an iterator according to the arguments.
`seq.tbltrain(tbl, trn)` produces an iterator over a table

An example syntax for `trn` is `'i1, i2, r1:r2', etc.` where `i1` and `i2` are individual indexes/elements, separated by `,` and `r1:r2` is a range (inclusive of end-point) denoted with a `:`. The range format follows python's numpy indexing, and a 'stride' can be given by including a second colon like `::2 -> is 1,3,5,...`, or `2::3 -> 2,5,8,...`. Negative numbers can be used to index relative to the length of the table, eg, `-1 -> len`, but if length is not given, negative indexing cannot be used and a number after the first colon must be provided. A missing left-number on the colon assumes `1`, and missing right number assumes `len`. A missing 'stride' (number after the optional second colon) assumes a value of 1.
Variable-like strings can be given in place of numbers, which are assumed to be keys for a table instead.
For `tbltrain` a `*` can be passed to iterate over all keys.

The default colon and comma separators for ranges and elements can be set with `seq.train_range_sep` and `seq.train_element_sep`, respectively.

```
\begin{luacode*}
  for i in
    pl.seq.itrain('1, :, 6, 0::2, -3 ',
                    5) do
      tex.print(i..',')
    end
  local t = {'n1','n2',a='A',b='B',c='C'}
    for k, v in
    pl.seq.tbltrain(t, '*,c,1') do
      tex.print(tostring(k)..'='..tostring(v)..'; ')
    end
\end{luacode*}
```

1, 1, 2, 3, 4, 5, 6, 0, 2, 4, 3, c=C; a=A; b=B; c=C; 1=n1;

A `pl.tex.` module is added

`add_bkt_cnt(n)`, `close_bkt_cnt(n)`, `reset_bkt_cnt` functions to keep track of adding curly brackets as strings. `add` will return `n` (default 1) {'s and increment a counter. `close` will return `n` }'s (default will close all brackets) and decrement.

`_NumBkts` internal integer for tracking the number of brackets

`opencmd(cs)` prints `\cs {` and adds to the bracket counters.

`openenv(env,opts)` prints a `\begin {env}[opts]`, and stores the enironment in a list so it can be later closed with `closeenv{num}`

`xNoValue,xTrue,xFalse`: `xparse` equivalents for commands

`prt(x),prtn(x)` print without or with a newline at end. Tries to help with special characters or numbers printing.

`prtl(l),prtt(t)` print a literal string, or table

`wrt(x), wrtn(x)` write to log

`wrth(s1, s2)` pretty-print something to console. S2 is a flag to help you find., alias is `help_wrt`, also in `pl.wrth`

`prt_array2d(tt)` pretty print a 2d array

`pkgwarn(pkg, msg1, msg2)` throw a package warning

`pkgerror(pkg, msg1, msg2, stop)` throw a package error. If stop is true, immediately ceases compile.

`defcmd(cs, val)` like `\gdef` , but note that no special chars allowed in `cs`(eg. `@`)

`defmacro(cs, val)` like `\gdef` , allows special characters, but any tokens in val must be predefined (this uses `token.set_macro` internally)

`newcmd(cs, val)` like `\newcommand`

`renewcmd(cs, val)` like `\renewcommand`

`prvcmd(cs, val)` like `\providecommand`

`deccmd(cs, dft, overwrite)` declare a command. If `dft` (default) is `nil`, `cs` is set to a package warning saying `'cs' was declared and used in document, but never set`. If

`overwrite` is true, it will overwrite an existing command (using `defcmd`), otherwise, it will throw error like `newcmd`.

`get_ref_info(l)` accesses the `\r @label` and returns a table

Recording LaTeX input as a lua variable

`penlight.tex.startrecording()` start recording input buffer without printing to latex
`penlight.tex.stoprecording()` stop recording input buffer
`penlight.tex.readbuf()` internal-use function that interprets the buffer. This will ignore an environment ending (eg. `end{envir}`)

`penlight.tex.recordedbuf` the string variable where the recorded buffer is stored

penlightplus LaTeX Macros

Macro helpers

`\MakeluastringCommands [def]{spec}` will let `\plluastring (A|B|C..)` be `\luastring (N|O|T|F)` based on the letters that `spec` is set to (or `def`(ault) if nothing is provided) This is useful if you want to write a command with flexibility on argument expansion. The user can specify `n`, `o`, `t`, and `f` (case insensitve) if they want none, once, twice, or full expansion.

Variants of luastring are added:
`\luastringF {m}` = `\luastring {m}`
`\luastringT {m}`, expand the first token of m twice

For example, we can control the expansion of args 2 and 3 with arg 1:

```
\NewDocumentCommand{\splittocomma}{ O{nn} m m }{%
   \MakeluastringCommands[nn]{#1}%
   \luadirect{penlight.tex.split2comma(\plluastringA{#2},\plluastringB{#3})}%
  }
```

Lua boolean expressions

`\ifluax {<Lua expr>}{<do if true>}[<do if false>]` and
`\ifluaxv {<Lua expr>}{<do if true>}[<do if false>]` for truthy (uses `penlight.hasval`). The argument is expanded.

```
\ifluax{3^3 == 27}{3*3*3 is 27}[WRONG]\\
\ifluax{abc123 == nil}{Var is nil}[WRONG]\\
\ifluax{not true}{tRuE}[fAlSe]\\
\ifluax{''}{TRUE}[FALSE]\\
\ifluaxv{''}{true}[false]\\
\def\XXX{8}
\ifluax{\XXX == 8}{Yes}[No]
```

3*3*3 is 27
Var is nil
fAlSe
TRUE
false
Yes

Case-switch for Conditionals

`\caseswitch {case}{kev-val choices}` The starred version will throw an error if the case is not found. Use `__` as a placeholder for a case that isn't matched. The case is fully expanded and interpreted as a lua string.

```
\def\caseswitchexample{\caseswitch{\mycase}{dog=DOG, cat=CAT, __=INVALID}}
\def\mycase{dog} \caseswitchexample \\
\def\mycase{human} \caseswitchexample
```

DOG
INVALID

PDF meta data (for pdfx package)

`\writePDFmetadatakv *[x]{kv}` Take a key-value string (eg. `title=whatever, author=me`) and then writes to the `jobname.xmpdata` file, which is used by pdfx. `*` will first clear `__PDFmetadata__` which is the table variable that stores the metadata. The un-starred version updates that table. You can control the expansion of the key-val argument with `[x]`, which is fully expanded by default. Command sequences are ultimately stripped from the values, except for `\and` is converted to `\sep` for pdfx usage (`https://texdoc.org/serve/pdfx/0`).

`\writePDFmetadata` runs the lua function `penlight.tex.writePDFmetadata()`, which pushes the lua variable `__PDFmetadata__` (a table) to the xmpdata file. This might be useful if you're updating `__PDFmetadata__` by some other means.

```
\writePDFmetadatakv{author=Some One} %
\writePDFmetadatakv*[n]{author=Kale \and You\xspace} % Overwrites above. Does not expant kv
\writePDFmetadatakv{date=2024-02-01}
```